

精通

NetBeans

Java 桌面、 Web 与企业级程序开发详解

吴亚峰 王鑫磊 编著

 **SUN** 中国技术社区
专家推荐
gceclub.sun.com.cn

全面阐述 Netbeans

通过实例讲解技术，深入浅出，上手容易

配有超值光盘



2 CD-ROM


本书源代码 + 多媒体视频

Sun 中国技术社区提供的资源盘

内容包括：

- NetBeans IDE 5.0
- NetBeans Mobility Pack for NetBeans IDE 5.0
- NetBeans Profiler for NetBeans IDE 5.0
- NetBeans 教程

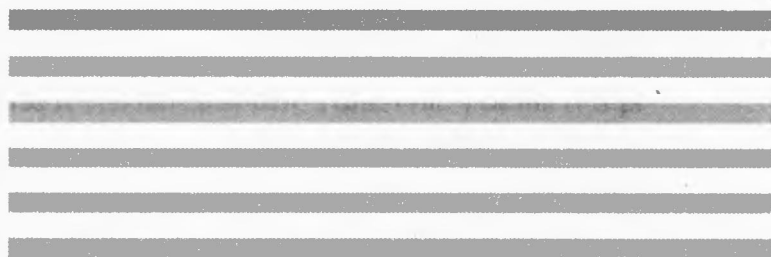
 **人民邮电出版社**
POSTS & TELECOM PRESS



NetBeans 作为后起之秀，以其独特、强大的功能迅速抢占着市场，吸引了大量 Java 开发者的注意。大家迫切希望能有一本全面介绍 NetBeans 使用的指导书。本书按照 Java 开发中的不同方向分类，比较完整地剖析了 NetBeans 的主要特点。希望通过这一个个开发中的实际场景，读者不仅可以学习到如何使用 NetBeans，更多的是全面地了解 Java 技术。

——叶亮，Sun 中国技术社区技术顾问

精通



NetBeans — Java 桌面、 Web 与企业级程序开发详解

吴亚峰 王鑫磊 编著

人民邮电出版社
北京

图书在版编目 (CIP) 数据

精通 NetBeans: Java 桌面、Web 与企业级程序开发详解 / 吴亚峰, 王鑫磊编著.

—北京: 人民邮电出版社, 2007.2

ISBN 978-7-115-13837-8

I. 精... II. ①吴...②王... III. JAVA 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字 (2006) 第 135033 号

内 容 提 要

NetBeans 是目前市场上流行的、使用方便的 Java 集成开发环境。本书结合丰富的程序实例讲述了 NetBeans 集成开发环境的使用, 读者在阅读本书的同时, 不仅能够掌握 NetBeans 的使用, 还能学习如何进行项目的开发。

本书共分为 5 篇, 第 1 篇是起步篇, 详细地讲述了 NetBeans 开发环境的各个功能组件。第 2 篇是 Swing/JFC 篇, 结合具体的实例对开发 Swing 程序用到的功能组件进行了介绍。第 3 篇为 Web 开发篇, 结合 Tomcat 详细地讲述了如何在 NetBeans 中进行 Web 程序开发。第 4 篇为 Java EE 篇, 结合具体的实例讲述了如何在 NetBeans 中开发企业级程序, 以及开发企业级程序需要用到的 Java 知识。第 5 篇为高级工具篇介绍了如何在 NetBeans 中集成已存在的 Ant 脚本, 如何使用 JUnit 进行单元测试, 如何使用 NetBeans Profiler 进行分析, 以及如何将已有的 Eclipse 项目导入到 NetBeans 中。

本书适用于具有一定开发经验的开发人员参考学习使用。

精通 NetBeans——Java 桌面、Web 与企业级 程序开发详解

◆ 编 著 吴亚峰 王鑫磊

责任编辑 屈艳莲

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号

邮编 100061 电子函件 315@ptpress.com.cn

网址 <http://www.ptpress.com.cn>

北京隆昌伟业印刷有限公司印刷

新华书店总店北京发行所经销

◆ 开本: 787×1092 1/16

印张: 38.5

字数: 931 千字

2007 年 2 月第 1 版

印数: 1—4 000 册

2007 年 2 月北京第 1 次印刷

ISBN 978-7-115-13837-8/TP · 4894

定价: 75.00 元 (附 2 张光盘)

读者服务热线: (010)67132692 印装质量热线: (010)67129223

序

光阴似剑，日月如梭，一晃 Java 已经诞生了十几年。在这十几年中，科技大大改变了我们的生活。现在，人们每天都在享受着高科技带来的方便、快捷的生活，而 Java 技术为此搭建了坚实的技术平台。

Java 语言是目前世界惟一可用于从智能卡应用、手持式电子消费类产品应用到桌面应用，以及企业级应用的平台。现在，Java 技术就意味着商机，全世界有 12.5 亿张 Java 智能卡、10 亿部 Java 手机以及 28 亿的 Java 设备在工作，就连发射到火星的探测器勇气号都是基于 Java 技术的。

随着 Java 技术与市场的不断发展，对 Java 开发工具的需求也与日俱增，NetBeans 就是其中一款非常优秀的 Java 开发工具。随着 NetBeans IDE 5.0 的发布，NetBeans 可以更加方便地管理 Java 开发的全过程。使用 NetBeans 可以方便地进行各个领域的 Java 开发，从简单的桌面应用到企业级应用，从手机嵌入式程序到智能卡。最吸引人的是，这款开发工具是完全开源的！

在这样技术日新月异、飞速发展的时代，作为一个 Java 开发人员，我们需要掌握最好用、最便捷的开发工具。《精通 NetBeans——Java 桌面、Web 与企业级程序开发详解》是开发人员深入学习 Java 开发的良师益友。

前 言

NetBeans 是 SUN 公司重点推广的 Java 集成开发环境, 随着 Java 语言的流行, 越来越多的开发者开始使用这个可视化的 Java 应用开发工具。使用 NetBeans 进行 Java 应用开发, 可以极大地提高新手的學習速度, 縮短熟手的开发时间, 这使得 NetBeans 成为广大开发人员最喜爱的开发工具之一。

NetBeans 简介

NetBeans 使用 Java 编程语言编写, 具有很好的可移植性。其强大的功能, 可以帮助开发人员编写、编译、调试和部署 Java 程序, 并将版本控制和 XML 编辑等众多功能融入其中。

NetBeans 提供了对开发 NetBeans 插件和基于 NetBeans 平台的富客户端全面的支持; 具备全新的 GUI 设计工具 Matisse, 崭新的 CVS 支持, 并且增强了编辑器的性能; 同时提供了对主流 J2EE 应用服务器的支持, 主要包括 Sun Application Server 8.2、Weblogic 9.0 和 JBoss 4。

NetBeans 集成了程序员开发桌面、Web、企业级和移动应用所需要的一切软件资源。下面列出了 NetBeans 的一些特性。

- 全面支持构建 NetBeans 插件模块和基于 NetBeans 平台的富客户端应用程序。
- 全面支持 SUN Application Server 8.2、Weblogic 9.0、JBoss 4 以及 Tomcat 5.x。
- 全面支持 Java 企业版应用程序消息服务。
- 增强了对 Java Server Faces 和 Struts 的支持。
- 提供了各种功能强大的代码编辑器。
- 加快了代码编译的速度。
- 重新实现了对 CVS 的支持。
- 改进的、用户界面友好的 IDE 外型。
- 可以通过 NetBeans Developer Collaboration 与其他开发者一起讨论和编辑代码。
- 集成了编写、测试以及调试 Java ME 程序的功能。

本书的形成

为帮助众多的开发人员熟练地使用 NetBeans 进行开发,使众多的初学者快速掌握开发 Java 程序的方法,作者根据多年的项目开发经验编写了本书,并依照学习规律组织了本书的内容,对于初学者和有一定工作经验的开发人员都有较大的帮助。

本书的特点

本书不仅详细地讲述了 NetBeans 的使用,并且讲述了 Java 的相关知识及编程思想。如在 Swing/JFC 篇中介绍了与 Swing 有关的知识,在 Web 开发篇中介绍了开发 Web 应用程序经常使用的 JSP、Servlet 与 JavaBean 技术,并且介绍了如何在 Tomcat 中部署及运行 Web 项目。

在 J2EE 篇中,不仅介绍了如何在 NetBeans 中开发 Java EE 程序,还介绍了如何安装、配置与使用 JBoss、MySQL、Weblogic 9.0 以及 Oracle10g。读者不仅学习了 NetBeans 的使用,而且掌握了众多与 Java 相关的技术。

本书在内容的编排以及目录的组织上都十分讲究,力求使读者能够循序渐进地掌握使用 NetBeans 进行 Java 程序开发的方法。对于每一个示例程序,都强调示例效果,以保证读者在实践的过程中不断地提高开发水平。

除此之外,本书还具有以下特点。

- ❶ 讲解通俗,步骤详细:本书以通俗易懂的语言讲解了相应的技术,以及如何在 NetBeans 中应用这些技术开发程序。读者只需按照步骤操作,就能体会到独立开发程序的乐趣。
- ❷ 实例经典,内容丰富:在本书中,每项技术都配有相应的实例,这样将会使读者更加容易理解书中所讲述的知识。丰富的内容涉及使用 NetBeans 进行项目开发的各个方面。
- ❸ 主次分明,结构清晰:本书在讲解实例的时候,基本思路是一致的,使读者能够更加轻松地阅读,快速地理解。
- ❹ 涉及面广,适应面宽:本书讲解了使用 NetBeans 进行项目开发的方方面面,无论是初学者还是有一定经验的开发人员,都能从本书中学习到很多知识。
- ❺ 配有光盘,辅助教学:本书配有多媒体光盘,书中的实例相关的文件和软件按照章节与实例名称分别存放。读者只需要按照书中介绍的步骤进行操作,即可得到一个完整功能的程序。

本书面向的读者

本书不仅介绍了 NetBeans,而且还介绍了开发 Java 程序时用到的多项技术,包括桌面版程序的开发、Web 程序的开发以及企业级程序的开发,深入浅出地讲解了如何在 NetBeans 中开发 Java 程序。因此本书适合所有 Java 初学者以及对 Java 有一定研究的开发人员。

- ❶ Java 初学者:阅读本书,既可以学习与 Java 有关的技术,又能够学习如何在 NetBeans 中开发项目。对于该类读者,本书绝对是一本从入门到精通的最好教材。
- ❷ 对 Java 有一定研究的开发人员:可以学到如何使用 NetBeans 进行项目开发以及如何提高程序的运行效率。书中的注意说明技巧与一些配置文件,绝对是你一生都事

用不尽的技术财富。

- 高级开发人员：即使你已经是一名使用 Java 进行程序开发的高手，仍然能够从本书中学习到很多的知识，比如如何进行项目的架构设计，如何提高整个项目的性能等。总之，对于各种类型的读者，都能从本书中找到需要的内容。

阅读本书的理由

本书是一本全面介绍 NetBeans 的书籍，它从软件开发者的角度出发来介绍 NetBeans 的使用。深入浅出地阐述了如何在 NetBeans 中开发 Java 程序。

本书中的大多数内容来源于作者开发实际项目的经验，实例也大部分的脱胎于实际的项目，同时针对本书的需要做出了相应的调整，使得本书更加符合广大读者的要求。

与其他书籍相比，本书还具有以下优点。

- 结合丰富具体的实例，讲述开发 Swing 程序需要用到的技术。
- 结合丰富具体的实例，讲述开发 Web 程序所需的技术。囊括了 JSP、Servlet 以及 JavaBean，并对 MVC 模式进行介绍。
- 结合丰富具体的实例，讲述开发 EJB 模块需要用到的技术，包括无状态会话 Bean、有状态会话 Bean、容器管理持久性的实体 Bean 以及消息驱动 Bean。
- 结合实例详细地介绍了主流的 Web、J2EE、数据库平台在 Java 中的应用。

关于作者

吴亚峰：毕业于北京邮电大学，后留学澳大利亚卧龙岗大学取得硕士学位，在 1998 年开始从事 Java 开发，有多年的 Java 开发经验，主要研究的方向为 Java EE，现在为 Java EE 独立软件工程师，同时兼任 Sun 授权 Java 培训中心认证教师。

王鑫磊：毕业于河北理工大学，从进入大学开始，就痴迷于编程，大二的时候就通过了 SCJP 认证；多年来一直致力于使用 Java 开发程序，曾在多个公司任职，现在主要研究持久层与 Web 层的应用开发，另外在数据库方面也有较深的研究。

致谢

本书在编写过程中得到了唐山百纳科技有限公司（SUN 授权 Java 培训中心）的大力支持，此外李迪锋和李贝为本书的编写提供了很多帮助，在此表示衷心感谢！

光盘使用说明

1. 工程环境

- Java 环境是 JDK1.5.0_06, IDE 环境是 NetBeans IDE 5.0
- 使用了 Tomcat 5.5.7 作为(不涉及 EJB 的)JSP/Servlet 容器,而 EJB 容器则采用了 JBoss

4.0.3

2. 工程目录

- build: 存放编译后的类文件
- dist: 存放打包后的文件,包括 jar 文件与 war 文件
- nbproject: 与 NetBeans 相关的配置文件
- src: 包含了所有的源文件
- test: 测试代码

3. 本书所附的源码

第 1 章 (\chapter01)

- \firstexample 第一个 NetBeans IDE 项目
- \idewindow 用来介绍 NetBeans 菜单的示例项目

第 2 章 (\chapter02)

- \addstudentinfo 用 NetBeans 开发的添加学生信息项目
- \src* 简单的 Swing 应用程序

第 4 章 (\chapter04)

- \container 使用 NetBeans 开发的与控件有关的项目

第 5 章 (\chapter05)

 \actionlist

使用 NetBeans 开发的动作列表项目

 \container

使用 NetBeans 开发的与控件有关的项目

第 6 章 (\chapter06)

 \component

使用 NetBeans 开发的与容器有关的项目

 \src*

使用 JApplet 的简单应用程序

第 7 章 (\chapter07)

 \layoutmanager

使用 NetBeans 开发的与布局管理器有关的项目

 \cardmanage

使用 NetBeans 开发的名片管理项目

 \src*

非 NetBeans 项目的程序

第 8 章 (\chapter08)

 \dialog

使用 NetBeans 开发的与对话框有关的项目

 \src*

输入对话框示例程序

第 9 章 (\chapter09)

 \menu

使用 NetBeans 开发的与菜单有关的项目

 \src*

包含本章中开发的所有非 NetBeans 项目的程序

第 10 章 (\chapter10)

 \image

使用 NetBeans 开发的与图形图像有关的项目

 \src*

包含本章中开发的所有非 NetBeans 项目的程序

 \gifEncoder*

gif 编码器的 jar 包

 \jspsmart*

包含文件上传组件的 jar 包

第 11 章 (\chapter11)

 \tree

使用 NetBeans 开发的与树有关的项目

 \src*

包含本章中开发的所有非 NetBeans 项目的程序

第 12 章 (\chapter12)

 \table

使用 NetBeans 开发的与表格有关的项目

 \src*

包含本章中开发的所有非 NetBeans 项目的程序

第 13 章 (\chapter13)

 \src*

存在 BUG 的 Swing 程序和改进后的 Swing 程序

第 14 章 (\chapter14)

■ \jsp&html*

测试 Tomcat 的简单程序

第 15 章 (\chapter15)

■ \DemoJspApplication

使用 NetBeans 开发的与 JSP 有关的项目

■ \jsp&html*

本章中开发的所有非 NetBeans 项目的 jsp 与 html 文件

第 16 章 (\chapter16)

■ \DemoServletApplication

使用 NetBeans 开发的与 Servlet 有关的项目

■ \jsp&html*

本章中开发的所有非 NetBeans 项目的 jsp 与 html 文件

■ \servlet*

本章中开发的所有非 NetBeans 项目的 Servlet 文件

■ \web.xml

本章中开发的所有非 NetBeans 项目的 Servlet 配置文件

第 17 章 (\chapter17)

■ \DemoJavaBeanApplication

使用 NetBeans 开发的与 JavaBean 有关的项目

■ \javabea**

本章中开发的所有非 NetBeans 项目的 JavaBean 文件

第 18 章 (\chapter18)

■ \Shop

使用 NetBeans 开发的网上商店项目

第 20 章 (\chapter20)

■ \StatelessBeanExample

用户消费信息的示例项目

第 21 章 (\chapter21)

■ \ShopCart

购物车示例项目

第 22 章 (\chapter22)

■ \BookShop

网上书店示例项目

第 23 章 (\chapter23)

■ \MDBeanExample

产品问题反馈示例项目

■ \MDBean*

非 NetBeans 项目的消息驱动 Bean 源文件

第 25 章 (\chapter25)

■ \JUnit_test*

用于单元测试的非 NetBeans 项目的程序

■ \NetBeansJUnit

用于单元测试的 NetBeans 项目

第 26 章 (\chapter26)

■ \ProfilerSwing

用于监控应用程序线程状态的 NetBeans 项目

- \ProfilerMethod 用于监控指定方法的 CPU 使用情况的 NetBeans 项目
- \ProfileMemoryLeaking 用于寻找内存漏洞的 NetBeans 项目

第 27 章 (\chapter27)

- \eclipse 被导入到 NetBeans 的 Eclipse 示例项目

选择 NetBeans 的理由

NetBeans 是一款非常方便易用的 IDE 开发环境，对 Java 各个方面的开发都有很好的支持，可以大大提高开发效率，其有很多令人激动的特性：

1. 项目是基于 Ant 组织

在 NetBeans 中项目是按照标准的 Ant 方式组织的，在使用 NetBeans 的过程中不仅可以通过 NetBeans 自身来管理项目，同样也可以使用 Ant 来对项目进行管理。由于 Ant 是非常流行的一个 Java 项目组织方式，这就使得 NetBeans 有了一个得天独厚的优势：使用 NetBeans 生成的 Ant 脚本同样可以在其他集成 Ant 的 IDE 或直接在命令行中运行，这对团队开发非常实用。

2. 更新中心功能

NetBeans 能够从 NetBeans 更新中心下载并安装更新包对自身进行更新。为了使中国地区的 NetBeans 用户更方便地使用这一功能，Sun 中国技术社区建立了一个 NetBeans 更新中心的本地镜像，其 URL 地址为：<http://gceclub.sun.com.cn/NetBeans>。

3. 编辑器方便使用

NetBeans 的代码编辑器非常方便使用，有着各种提高代码编写效率的措施。这些措施包括非常方便并可以自定义的代码缩写功能，可以同步显示 API 的代码自动补全功能，而且还有非常聪明的代码自动修正功能，可以实现代码错误的“一键修复”。

4. 齐全的项目模板及实例

用 NetBeans 创建项目你的感觉应该是站在巨人的肩膀上，NetBeans 自带了种类丰富的项目模板，可以方便快捷地创建各种类型的项目。同时，项目模板中还包含 Java Blueprints 的很多项目，方便通过 NetBeans 学习 Java。

5. 支持使用 Matisse 进行 GUI 用户界面设计

使用 NetBeans 设计 GUI 用户界面将不再是一件令人厌烦的工作，而是一件轻松愉快的享受。使用 Matisse 以及 NetBeans 支持的 GroupLayout 布局管理器可以通过鼠标的拖曳方便地完成复杂界面的设计。

另外，GroupLayout 布局管理器对复杂界面的设计是一个福音，使用 GroupLayout 布局管理器管理器后，控件的大小和位置之间有了逻辑关系。比如，你可以告诉界面“a 按钮永远与 b 文本框对齐”、“c 标签与 d 标签上下相距 50 像素”等，这样进行界面设计是多么令人愉快的工作。

6. 方便的 Web 开发与调试

NetBeans 中集成了 Tomcat 5.5.9，用户如果没有特别需要可以直接基于 NetBeans 自带的 Tomcat 进行 Web 程序的开发与调试。当然，如果愿意也可以安装自己的 Tomcat 并使用 NetBeans 进行管理。

同时，NetBeans 还集成支持现在最流行的 Struts 和 JSF Web 层框架，在 NetBeans 中可以方便地进行这两种框架的开发。

6. 便捷的 J2EE 开发与调试

NetBeans 支持现在大部分主流的 J2EE 应用服务器，包括 SUN 的 Application Server、BEA 的 WebLogic 9.0、JBoss 组织的 JBoss 4.0 等，可以方便地基于这些 EJB 容器进行集成开发与调试。

使用 NetBeans 不仅可以方便地进行传统的 EJB 的开发，而且也可以很方便地进行 J2EE 其他方面的开发以及资源的管理，部署与调试等。

7. 强大的插件扩展功能

NetBeans 与 Eclipse 一样，也很方便地支持插件的开发。通过 NetBeans 中的插件以及插件套装项目模板可以很方便的进行单个插件以及插件套件的开发。

8. 易学易用的测试与调试功能

NetBeans 集成支持使用 JUnit 生成测试，对各种类型的 Java 项目进行方便地测试，省去了自己手工编写单元测试用例的麻烦。同时，通过使用 NetBeans Profiler 可以方便地监控与分析本地与远程程序的运行情况，方便了解程序运行过程中各方面的信息，简化程序的分析与调试过程。

9. 移动开发的利器

NetBeans 对移动开发有着非常完善的支持，通过使用 NetBeans 移动套件进行移动开发将大大简化开发流程，大部分令人厌烦的重复性编程工作都可以在鼠标的单击与拖曳中烟消云散，开发人员可以更加集中精力进行业务逻辑的开发。

目 录

第1篇 起步篇

第1章 NetBeans 集成开发环境	3
1.1 NetBeans 开发环境的搭建	3
1.1.1 JDK 的安装	3
1.1.2 JDK 的卸载	5
1.1.3 NetBeans 的安装	5
1.1.4 NetBeans 的卸载	7
1.2 NetBeans 的更新	7
1.3 创建第一个 NetBeans 项目	8
1.4 NetBeans 中的项目模板	10
1.5 NetBeans 示例项目的使用	10
1.6 NetBeans 开发环境组件介绍	11
1.6.1 主菜单	11
1.6.2 主环境	16
1.6.3 项目窗口	17
1.6.4 文件窗口	18
1.6.5 运行环境窗口	19
1.6.6 导航窗口	20
1.6.7 对象观察器窗口	20
1.6.8 源代码编辑器	20
1.6.9 GUI 设计器	22
1.6.10 属性窗口	22
1.7 NetBeans 常用的快捷键	23

1.7.1 基本常用快捷键	23
1.7.2 缩写展开快捷键	24
1.7.3 代码自动修复快捷键	25
1.8 自定义 NetBeans 开发环境	26
1.9 小结	28

第2篇 GUI 程序设计: Swing/JFC 篇

第2章 Swing 概述	31
2.1 初识 Swing	31
2.1.1 什么是 Swing	31
2.1.2 一个简单的 Swing 程序	32
2.2 Swing 的特性	34
2.2.1 轻量级控件	34
2.2.2 多种外观风格	34
2.2.3 更多的附加特性	34
2.3 Swing 与 AWT	35
2.4 使用 NetBeans 开发 Swing 应用程序	35
2.5 小结	38
第3章 Swing/JFC 的事件模型基础	39
3.1 概述	39
3.2 AWT 与 Swing 事件	40
3.3 动作事件	43
3.4 键盘事件	43
3.5 鼠标事件	43
3.6 窗口事件	43
3.7 事件适配器	43
3.8 小结	45
第4章 Swing 常用基本控件	47
4.1 Swing 控件类	47
4.2 Swing 标签	50
4.2.1 JLabel 类简介	50
4.2.2 在 JLabel 中使用图像	52
4.2.3 在 NetBeans 中使用 JLabel	52
4.3 Swing 按钮	54
4.3.1 JButton 类简介	54

4.3.2 NetBeans 中 JButton 的使用	55
4.4 Swing 文本框	57
4.4.1 JTextField 类简介	57
4.4.2 NetBeans 中 JTextField 的使用	58
4.5 格式化文本框	60
4.5.1 JFormattedTextField 类简介	60
4.5.2 NetBeans 中 JFormattedTextField 的使用	62
4.6 Swing 文本区	65
4.6.1 JTextArea 类简介	65
4.6.2 NetBeans 中 JTextArea 的使用	67
4.7 单选按钮与复选框	69
4.7.1 JCheckBox 类简介	69
4.7.2 JRadioButton 类简介	70
4.7.3 开发使用 JCheckBox 和 JRadioButton 的界面	71
4.7.4 开发业务代码	72
4.7.5 程序功能演示	75
4.8 编辑器面板	76
4.8.1 JEditorPane 类简介	76
4.8.2 NetBeans 中 JEditorPane 的使用	78
4.9 小结	80
第 5 章 Swing 常用高级控件	81
5.1 Swing 列表	81
5.1.1 JList 类简介	81
5.1.2 控件 MVC 思想简介	83
5.1.3 NetBeans 中开发使用 JList 程序	84
5.1.4 程序功能演示	86
5.2 Swing 组合框	87
5.2.1 JComboBox 类简介	87
5.2.2 NetBeans 中 JComboBox 的使用	89
5.3 Swing 分割窗口	90
5.3.1 JSplitPane 类简介	91
5.3.2 NetBeans 中 JSplitPane 的使用	92
5.4 实战: 使用 NetBeans 构建 Swing 的综合例程	93
5.4.1 项目概述	93
5.4.2 界面设计	94
5.4.3 功能代码的开发	95
5.5 小结	98

第 6 章 Swing 容器类	99
6.1 Swing 框架——JFrame	99
6.1.1 JFrame 类简介	99
6.1.2 NetBeans 中 JFrame 的使用	101
6.2 Swing 窗口——JWindow	104
6.2.1 JWindow 类简介	104
6.2.2 在 NetBeans 中使用 JWindow	105
6.3 小应用程序——JApplet	108
6.3.1 JApplet 简介	108
6.3.2 Applet 的标记和属性	109
6.3.3 Applet 的安全机制	110
6.3.4 Applet 的生命周期	111
6.3.5 NetBeans 中 JApplet 的使用	112
6.4 小结	115
第 7 章 布局管理器	117
7.1 为什么使用布局管理器	117
7.2 Java 中的布局管理器	118
7.3 流布局	118
7.3.1 FlowLayout 简介	118
7.3.2 在 NetBeans 中设置 FlowLayout 布局	120
7.4 边框布局	120
7.4.1 BorderLayout 简介	121
7.4.2 在 NetBeans 中设置 BorderLayout 布局	123
7.5 网格布局	124
7.5.1 GridLayout 简介	124
7.5.2 在 NetBeans 中设置 GridLayout 布局	125
7.6 卡片布局	128
7.6.1 CardLayout 简介	128
7.6.2 在 NetBeans 中使用 CardLayout 布局	131
7.7 箱式布局	132
7.7.1 BoxLayout 简介	132
7.7.2 使用 Box 类	134
7.7.3 Box 容器的嵌套使用	137
7.7.4 在 NetBeans 中使用 BoxLayout 布局	138
7.8 使用 GroupLayout 布局管理器	139
7.8.1 GroupLayout 简介	139
7.8.2 使用 GroupLayout 布局管理器设置间距	139

7.8.3 使用 GroupLayout 布局管理器设置控件的对齐	140
7.8.4 使用 GroupLayout 布局管理器进行“粘贴”	141
7.9 空布局	142
7.9.1 null 布局简介	143
7.9.2 在 NetBeans 中使用 null 布局	143
7.10 用 NetBeans 开发使用复合布局管理器的程序	144
7.10.1 项目概述	144
7.10.2 界面设计	145
7.10.3 功能代码的开发	148
7.10.4 项目总结	151
7.11 小结	151
第 8 章 Swing 对话框	153
8.1 Swing 对话框类——JDialog	153
8.1.1 JDialog 类简介	153
8.1.2 在 NetBeans 中使用 JDialog	154
8.2 使用 JOptionPane	156
8.2.1 JOptionPane 类简介	156
8.2.2 JOptionPane 的 4 种对话框	158
8.3 文件选择器	161
8.4 颜色选择器	163
8.4.1 JColorChooser 类简介	163
8.4.2 在 NetBeans 中开发 JColorChooser 的程序	165
8.5 实战：在 NetBeans 中开发复合对话框的程序	167
8.5.1 项目概述	167
8.5.2 界面设计	168
8.5.3 功能代码的开发	169
8.5.4 项目总结	172
8.6 小结	172
第 9 章 Swing 菜单	173
9.1 Swing 菜单简介	173
9.1.1 Swing 菜单控件说明	173
9.1.2 一个简单的菜单程序	174
9.2 菜单栏——JMenuBar	176
9.2.1 JMenuBar 简介	176
9.2.2 在 NetBeans 中使用 JMenuBar	177
9.3 菜单项——JMenuItem	178
9.3.1 菜单项简介	178

9.3.2 为菜单项添加快捷键和加速器	179
9.4 菜单——JMenu	180
9.5 在 NetBeans 中使用菜单控件	182
9.6 单选与复选菜单项	184
9.6.1 复选菜单项——JCheckBoxMenuItem	184
9.6.2 单选菜单项——JRadioButtonMenuItem	185
9.6.3 在 NetBeans 中使用单选与复选菜单项	186
9.7 弹出式菜单——JPopupMenu	188
9.7.1 JPopupMenu 类简介	188
9.7.2 在 NetBeans 中使用 JPopupMenu	191
9.8 小结	194
第 10 章 Java 与图像处理	195
10.1 图像类 Image	195
10.1.1 Image 类简介	195
10.1.2 在 Java 应用程序中绘制图像	196
10.1.3 设置 Java 应用程序窗口的图标	198
10.2 Swing 图标 ImageIcon	199
10.2.1 Icon 接口	199
10.2.2 ImageIcon 类	201
10.3 图像处理的高级应用——JPEG 编码器	204
10.4 其他编码器介绍	204
10.4.1 GifEncoder 简介	205
10.4.2 使用 GifEncoder 编码器程序的界面设计	205
10.4.3 使用 GifEncoder 编码器程序的功能代码的开发	206
10.5 在 NetBeans 中进行图像处理的案例	209
10.5.1 界面设计	209
10.5.2 功能代码的开发	209
10.6 小结	212
第 11 章 树状列表——JTree	213
11.1 与树有关的一些概念	213
11.2 JTree 介绍	214
11.2.1 JTree 类构造器说明	214
11.2.2 JTree 类的方法说明	215
11.2.3 使用 JTree 的程序实例	217
11.3 默认树模型	218
11.3.1 DefaultTreeModel 类构造器介绍	219
11.3.2 DefaultTreeModel 的常用方法说明	219

11.4 默认树节点.....	221
11.4.1 DefaultMutableTreeNode 类构造器说明.....	221
11.4.2 DefaultMutableTreeNode 类的获取、添加及修改方法.....	221
11.4.3 DefaultMutableTreeNode 类的枚举方法.....	224
11.5 树路径.....	224
11.5.1 TreePath 类的构造器.....	224
11.5.2 TreePath 类的常用方法说明.....	225
11.6 树的事件.....	226
11.6.1 选择事件.....	226
11.6.2 扩展事件.....	226
11.6.3 模型结构变化事件.....	227
11.7 树单元绘制器.....	227
11.7.1 默认树单元绘制器.....	227
11.7.2 自定义单元绘制器.....	229
11.8 在 NetBeans 中开发树状结构的程序实例.....	230
11.8.1 项目概述.....	230
11.8.2 界面设计.....	230
11.8.3 初始化操作及处理节点的选择, 展开及更改事件.....	231
11.8.4 增加与删除节点.....	234
11.8.5 更改节点图标.....	236
11.8.6 项目总结.....	238
11.9 小结.....	238
第 12 章 表格的高级应用.....	239
12.1 一个使用表格的简单程序.....	239
12.2 JTable 控件介绍.....	240
12.2.1 JTable 类简介.....	240
12.2.2 NetBeans 中 JTable 的使用实例.....	245
12.3 表格模型——TableModel.....	248
12.3.1 AbstractTableModel 类.....	248
12.3.2 DefaultTableModel 类.....	251
12.3.3 在 NetBeans 中开发使用表格模型的程序.....	253
12.4 表格列——TableColumn.....	256
12.5 表格列模型.....	257
12.5.1 默认表格列模型.....	258
12.5.2 默认表格列模型的常用方法说明.....	258
12.5.3 一个使用表格列模型的简单程序.....	259
12.6 与表格有关的事件.....	261
12.6.1 表格常用事件及处理方法.....	261

12.6.2 在 NetBeans 中开发处理表格事件的程序实例	262
12.7 表格编辑器与绘制器	264
12.7.1 表格绘制器简介	264
12.7.2 表格编辑器简介	266
12.7.3 在 NetBeans 中开发自定义表格编辑/绘制器的程序	267
12.8 小结	271
第 13 章 Swing 线程	273
13.1 一个存在 BUG 的简单程序	273
13.2 Swing 线程的工作原理	275
13.2.1 事件分发线程工作原理	275
13.2.2 事件分发线程模型引发的问题	275
13.3 解决 Swing 单线程问题	276
13.4 小结	278

第 3 篇 Web 开发篇

第 14 章 Tomcat 配置及应用	281
14.1 Tomcat 简介	281
14.1.1 Web 程序介绍	281
14.1.2 Tomcat 与传统的 Web 服务器	282
14.1.3 Tomcat 的 Realm 支持	283
14.1.4 Tomcat 与 J2EE 服务器	283
14.2 安装与配置 Tomcat	283
14.2.1 安装 Tomcat	284
14.2.2 测试 Tomcat 是否可用	285
14.2.3 开发并部署第一个 JSP	287
14.3 在 NetBeans 中使用 Tomcat	288
14.3.1 向 NetBeans 中添加 Tomcat	288
14.3.2 配置 Tomcat	289
14.3.3 操作 Tomcat	290
14.3.4 设置 NetBeans 使用的默认浏览器	291
14.4 其他 Web 服务器简介	291
14.5 小结	292
第 15 章 JSP 技术及应用	293
15.1 JSP 概述	293

15.1.1 JSP 技术介绍	293
15.1.2 ASP 与 PHP	294
15.1.3 JSP 的优势	295
15.2 在 NetBeans 中使用 JSP	295
15.2.1 在 NetBeans 中创建 Web 项目	296
15.2.2 在 NetBeans 中配置 Web 项目	297
15.2.3 在 NetBeans 中开发 JSP 程序	299
15.3 JSP 的模板元素	301
15.4 JSP 的脚本元素	301
15.5 JSP 注释	303
15.6 JSP 的指令元素	305
15.6.1 页面指令	305
15.6.2 include 指令	307
15.6.3 taglib 指令	309
15.7 JSP 动作元素	310
15.7.1 <jsp:param>与<jsp:include>动作元素	310
15.7.2 <jsp:forward>动作元素	312
15.8 常用的 JSP 内建对象	314
15.8.1 输出内建对象——out	315
15.8.2 请求内建对象——request	316
15.8.3 响应内建对象——response	318
15.8.4 会话内建对象——session	319
15.9 在 NetBeans 中开发完整的 JSP 程序	321
15.9.1 添加 HTML 文件	321
15.9.2 向 HTML 页面中添加控件	322
15.9.3 开发处理表单数据的 JSP 程序	325
15.10 小结	326
第 16 章 Servlet 技术及应用	327
16.1 Servlet 技术简介	327
16.2 开发并部署一个简单的 Servlet	328
16.3 在 NetBeans 中开发与配置 Servlet	330
16.3.1 在 NetBeans 中开发 Servlet	330
16.3.2 在 NetBeans 中配置及运行 Servlet	333
16.4 与 Servlet 有关的常用接口和类	333
16.4.1 Servlet 实现相关	334
16.4.2 Servlet 配置相关	334
16.4.3 Servlet 异常相关	335
16.4.4 请求与响应相关	336

16.4.5	HttpServletRequest 接口	336
16.4.6	HttpServletResponse 接口	339
16.4.7	会话相关	340
16.4.8	开发用来提交与处理信息的页面	341
16.4.9	Servlet 上下文	343
16.4.10	RequestDispatcher 接口	343
16.5	使用 HttpServlet 处理客户端请求	344
16.5.1	doGet 方法介绍	344
16.5.2	doPost 方法介绍	346
16.5.3	其他方法介绍	348
16.6	配置 Servlet	348
16.6.1	演示程序	349
16.6.2	Servlet 的名称、类以及其他选项	350
16.6.3	Servlet 的初始化参数	350
16.6.4	启动和加载优先级	350
16.6.5	Servlet 的映射	351
16.7	在 NetBeans 使用 Servlet 实现文件上传	352
16.7.1	开发实现文件上传的 Servlet	352
16.7.2	开发 JSP 程序	353
16.8	小结	354
第 17 章	JavaBean 组件模型	355
17.1	JavaBean 简介	355
17.2	与 JavaBean 有关的概念	355
17.2.1	JavaBean 的属性简介	356
17.2.2	简单属性	356
17.2.3	索引属性	357
17.2.4	绑定属性	358
17.2.5	约束属性	358
17.2.6	JavaBean 的方法	359
17.3	在 JSP 中使用 JavaBean	359
17.3.1	<jsp:useBean>动作指令简介	359
17.3.2	<jsp:setProperty>与<jsp:getProperty>简介	360
17.4	使用 JavaBean 封装数据库连接	361
17.4.1	创建数据库和表	362
17.4.2	配置 ODBC 数据源	363
17.4.3	开发表示用户信息的 JavaBean 组件	364
17.4.4	开发封装数据库操作的 JavaBean 组件	365
17.4.5	开发 HTML 页面与 JSP 页面	367

17.5 用 JavaBean 实现购物车	369
17.6 在 Servlet 中使用 JavaBean	371
17.6.1 <jsp:useBean>标签与 Servlet	372
17.6.2 开发 JavaBean	373
17.6.3 开发 Servlet	373
17.6.4 开发 HTML 与 JSP	373
17.7 HTTP 监视器	375
17.7.1 设置 HTTP 监视器	375
17.7.2 分析 HTTP 请求	376
17.7.3 重现 HTTP 请求	377
17.8 小结	378
第 18 章 开发 MVC 架构的网上商店	379
18.1 MVC 架构介绍	379
18.1.1 JSP 的 Model 1 与 Model 2 架构	379
18.1.2 MVC 模式简介	380
18.2 项目功能演示	382
18.2.1 查询功能演示	382
18.2.2 购买功能演示	383
18.2.3 管理功能演示	384
18.3 模块功能及相互关系说明	385
18.3.1 查询购买模块	386
18.3.2 管理模块	387
18.4 查询购买模块的开发	387
18.4.1 数据库准备工作	388
18.4.2 开发 ProductBean、CartProduct 与 CartBean	388
18.4.3 开发 DataBaseBean	390
18.4.4 开发查询购买模块的 Servlet	392
18.4.5 开发查询与显示页面	394
18.4.6 开发购物车页面与结算页面	397
18.4.7 开发欢迎页面	400
18.5 开发管理模块	400
18.5.1 创建数据库表	400
18.5.2 开发 JavaBean	400
18.5.3 开发 Servlet	403
18.5.4 开发显示与删除页面	405
18.5.5 开发登录页面与添加商品页面	407
18.6 小结	409

第4篇 Java EE 篇

第19章 NetBeans 与 Java EE	413
19.1 Java EE 概述	413
19.2 Enterprise JavaBeans 概述	413
19.2.1 EJB 简介	414
19.2.2 EJB 的优点及适用场合	415
19.3 NetBeans 对 Java EE 的支持	415
19.4 MySQL 数据库	416
19.4.1 安装 MySQL 数据库	416
19.4.2 配置 MySQL 数据库	418
19.4.3 连接 MySQL 数据库	419
19.4.4 管理 MySQL 数据库	420
19.5 在 NetBeans 中连接 Oracle10g	423
19.5.1 安装 Oracle10g	423
19.5.2 配置 Oracle10g	425
19.5.3 测试 Oracle10g	426
19.5.4 NetBeans 连接 Oracle10g	426
19.5.5 管理 Oracle 数据库	427
19.6 在 NetBeans 中配置 Weblogic 9.0	428
19.6.1 设置 Weblogic 9.0 的安装类型与目录	428
19.6.2 设置其他安装选项并安装 Weblogic 9.0	429
19.6.3 配置 Weblogic 9.0	431
19.6.4 测试 Weblogic 9.0	433
19.6.5 连接 Weblogic 9.0 服务器	433
19.6.6 使用 Weblogic 9.0 服务器	434
19.7 配置 JBoss4.0	435
19.7.1 安装与测试 JBoss 4.0	435
19.7.2 在 NetBeans 中配置 JBoss 4.0	436
19.8 在 NetBeans 中管理 DTD/XML 库	437
19.9 小结	438
第20章 无状态会话 Bean——用户消费信息登记	439
20.1 无状态会话 Bean 简介	439
20.2 项目简介	440
20.2.1 项目功能演示	440
20.2.2 模块结构介绍	441
20.3 建立数据库表	442

20.4 创建项目	443
20.5 开发 EJB 模块	444
20.5.1 开发 EJB 代码框架	444
20.5.2 业务代码的开发	446
20.5.3 配置 EJB 模块	448
20.6 开发 Web 模块	449
20.6.1 开发用户输入页面与信息输入成功页面	449
20.6.2 开发信息输入错误页面、查询页面	452
20.6.3 开发查询结果页面	453
20.6.4 控制器 Servlet 的开发	454
20.6.5 向类路径中添加并配置 EJB 模块	457
20.7 编译运行整个项目	458
20.8 小结	458
第 21 章 有状态会话 Bean——实现购物车	459
21.1 有状态会话 Bean 简介	459
21.2 项目功能简介	459
21.3 开发 EJB 模块	461
21.3.1 有状态会话 Bean 的创建与依赖值类的开发	461
21.3.2 业务代码的开发	463
21.4 开发 Web 模块	465
21.4.1 控制器 Servlet 的开发	465
21.4.2 开发商品的显示与购买页面	467
21.4.3 开发购物车	471
21.5 编译运行整个项目	473
21.6 小结	474
第 22 章 开发 CMP 实体 Bean——图书信息管理	475
22.1 CMP 简介	475
22.1.1 实体 Bean 的简单介绍	475
22.1.2 CMP 的简单介绍	476
22.1.3 持久性类的简单介绍	476
22.2 项目功能介绍	476
22.3 项目的结构及模块功能说明	478
22.4 数据库准备工作	479
22.5 开发 CMP 模块	480
22.5.1 CMP 的创建与关系字段的添加	481
22.5.2 添加 create 与 find 方法	483
22.5.3 配置主键与关系字段	484

22.5.4 配置 CMP 与数据库映射	485
22.6 开发 Web 模块	488
22.6.1 控制器 Servlet 的开发	488
22.6.2 开发 JSP	491
22.7 编译运行整个项目	494
22.8 小结	494
第 23 章 消息驱动 Bean——商品问题反馈系统	495
23.1 Java 消息服务	495
23.1.1 JMS 简介	495
23.1.2 JMS 的异步性	496
23.1.3 JMS 消息模型	496
23.2 基于 JMS 的消息驱动 Bean	496
23.3 开发一个简单的使用消息驱动 Bean 的程序	497
23.3.1 项目功能简介	497
23.3.2 配置消息服务	498
23.3.3 开发、配置与部署消息驱动 Bean	499
23.3.4 开发与配置控制器 Servlet	502
23.3.5 开发 JSP	504
23.3.6 运行项目	506
23.4 在 NetBeans 中开发消息驱动 Bean	506
23.4.1 项目功能简介	506
23.4.2 准备工作	507
23.4.3 开发与配置消息驱动 Bean	508
23.4.4 控制器 Servlet 的开发	510
23.4.5 开发 JSP	512
23.4.6 编译并运行项目	513
23.5 小结	514

第 5 篇 高级工具篇

第 24 章 使用 NetBeans 集成已存在的 Ant 脚本	517
24.1 在 NetBeans 中使用 Ant 脚本	517
24.1.1 在 NetBeans 中使用 Ant 脚本的原因	517
24.1.2 NetBeans 的项目系统与自由格式项目的区别	518
24.1.3 在 NetBeans 中使用 Ant 脚本的步骤	518
24.2 创建自由格式项目	518
24.3 为项目设置命令	521
24.3.1 将目标映射到 NetBeans 命令	521

24.3.2 为 Java 应用程序设置 Debug Project 命令	522
24.3.3 为 Web 应用程序设置 Debug Project 命令	524
24.4 为文件设置命令	526
24.4.1 为文件设置命令的说明	526
24.4.2 设置 Compile File 命令	526
24.4.3 设置 Run File 命令	528
24.4.4 设置 Debug File 命令	529
24.5 更改自由格式项目的目标 JDK	530
24.6 小结	531
第 25 章 在 NetBeans 中使用 JUnit	533
25.1 什么是单元测试	533
25.2 JUnit 简介	533
25.3 TestCase 类简介	534
25.4 TestSuite 类简介	537
25.5 在 NetBean 中使用 JUnit 进行测试	539
25.5.1 创建测试类	539
25.5.2 查看自动生成的代码	541
25.5.3 修改并运行测试程序	544
25.5.4 在 NetBeans 中开发测试套件	545
25.6 小结	548
第 26 章 NetBeans Profiler: 监控应用程序的执行	549
26.1 NetBeans Profiler 简介	549
26.2 NetBeans Profiler 安装配置	549
26.3 监控 Swing 应用中的线程状态	550
26.3.1 运行示例项目	550
26.3.2 监控线程状态	551
26.4 监控指定方法的 CPU 使用情况	554
26.4.1 运行示例程序	554
26.4.2 选择测试的根方法	555
26.4.3 监控程序的运行	556
26.5 通过 NetBeans Profiler 寻找内存漏洞	558
26.6 小结	561
第 27 章 将 Eclipse 项目导入 NetBeans	563
27.1 概述	563
27.2 获取与安装 Eclipse 项目导入器	563
27.3 导入 Eclipse 项目	566

27.3.1 两种导入方式的说明	566
27.3.2 导入 Eclipse 项目并保存项目依赖关系	567
27.3.3 导入 Eclipse 项目并忽略依赖关系	569
27.4 小结	570
附录 A NetBeans 5.0 的下载过程	571
附录 B JDK 的简单使用	573
附录 C NetBeans 常用菜单项的中英文对照表	577
附录 D 术语表	581

本篇是学习使用 NetBeans 进行项目开发的基础，主要介绍了如下内容。

- 如何安装 NetBeans 开发平台；
- 如何使用 NetBeans 示例项目；
- 如何使用 NetBeans 的更新中心；
- 简单的示例；
- 开发环境组件介绍，包括主环境、项目窗口和文件窗口等内容；
- 常用快捷键介绍；
- 自定义 NetBeans 的开发环境。

第 1 篇 起步篇

第 1 章 NetBeans 集成开发环境

第 1 章

NetBeans 集成开发环境

本章将首先介绍 NetBeans 开发环境的搭建和 NetBeans 的更新, 然后开发一个简单的例子, 使读者对 NetBeans 5.0 有进一步的了解。接着对 NetBeans 5.0 的常用组件及其快捷键进行了详细的介绍。在本章最后将介绍如何自定义 NetBeans 5.0 的开发环境。

1.1 NetBeans 开发环境的搭建

学习 NetBeans 5.0 之前, 首先需要搭建 NetBeans 5.0 的开发环境。本节旨在为读者介绍如何在各种不同的操作系统下面搭建 NetBeans 5.0, 有经验的读者可以略过此节。

1.1.1 JDK 的安装

安装 NetBeans 5.0 之前, 首先需要安装 JDK, 当前最新版本为 1.5.0_06, 根据不同的平台, 可以从下面所列的地址下载相应的 JDK。

- Windows: <http://java.sun.com/j2se/1.5.0/download.html>;
- Solaris: <http://java.sun.com/j2se/1.5.0/download.html>;
- Linux: <http://java.sun.com/j2se/1.5.0/download.html>;
- Mac OS X: <http://www.apple.com/support/downloads/javaupdate142.html>;
- Open VMS: <http://h18012.www1.hp.com/java/download/index.html>。

下载完成后, 即可进行安装。下面介绍在 Windows 操作系统下安装 JDK 的方法。

(1) 进入存放 Java SDK 1.5 软件包的目录, 双击 `jdk-1_5_0_06-windows-i586-p.exe`, 运行 Java SDK 1.5 的安装程序。

(2) 在经过短暂的初始化工作以后, 安装界面如图 1-1 所示。

(3) 选中“我接受许可证协议中的条款”单选按钮, 单击“下一步”按钮继续安装, 此时安装界面如图 1-2 所示。

(4) 在此处选择要安装的功能组件, 单击“更改”按钮选择安装目录。完成设置后, 单击“下一步”按钮继续安装, 安装界面如图 1-3 所示。

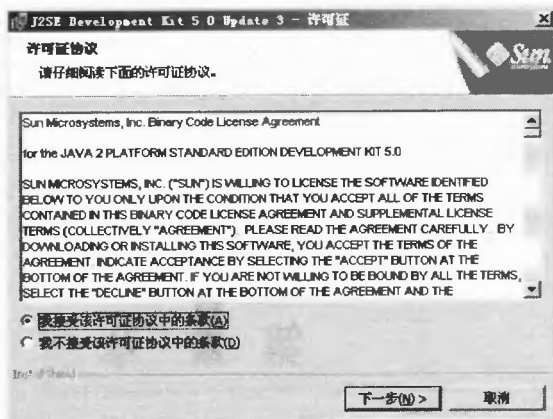


图 1-1 接受许可协议

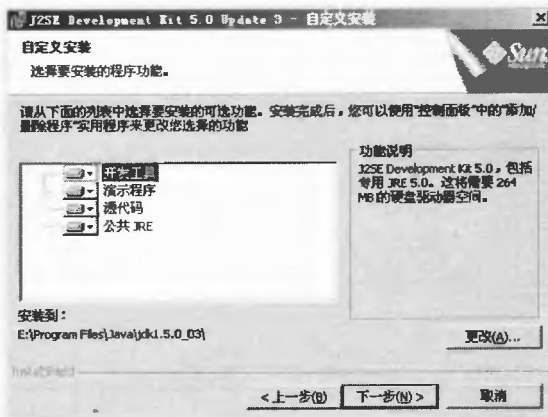


图 1-2 选择安装功能与目录

(5) 在安装完功能组件后，会弹出如图 1-4 所示的窗口，在此处配置 Java 运行时环境 (JRE) 安装。

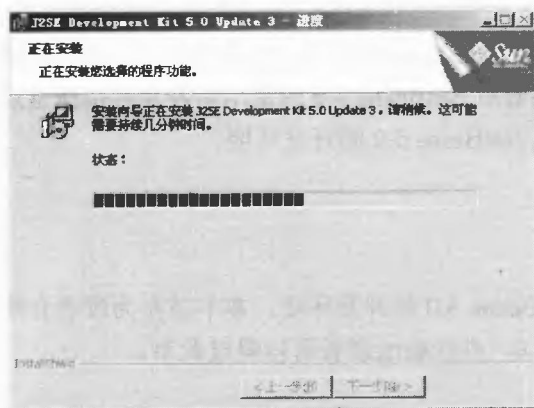


图 1-3 安装进度窗口

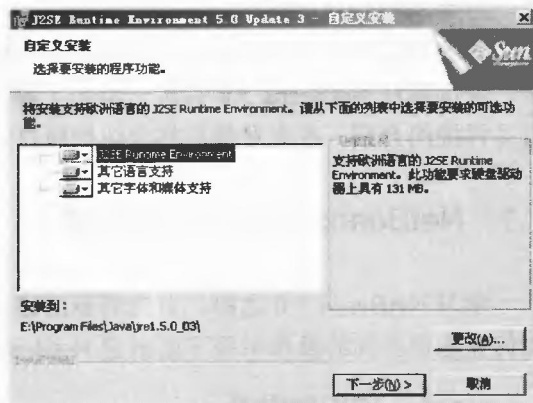


图 1-4 安装 JRE

(6) 选择安装语言支持等功能，并单击“更改”按钮选择安装目录。完成设置后，单击“下一步”按钮，程序进入如图 1-5 所示的界面。

(7) 在此可以选择要注册 Java (TM) 插件的浏览器，完成设置以后，单击“下一步”按钮，继续安装。等待一段时间以后，出现如图 1-6 所示的窗口。

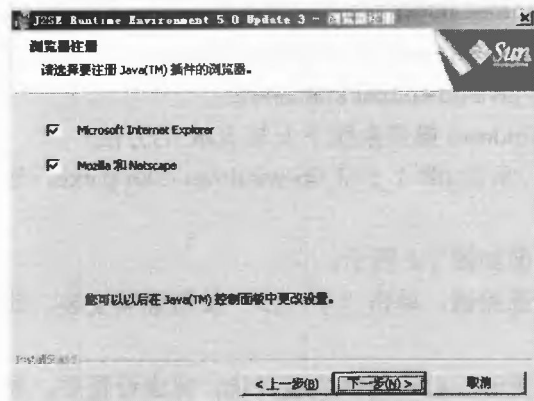


图 1-5 注册 Java (TM) 插件浏览器



图 1-6 完成安装

(8) 在如图 1-6 所示的窗口中单击“完成”按钮退出。

在其他操作系统 (Linux、Solaris) 中安装 Java SDK 1.5 的过程与在 Windows 下的安装过程基本相同, 读者可以参照 Windows 下的安装过程自行练习。

1.1.2 JDK 的卸载

下面介绍如何从 Windows 操作系统中卸载 J2SE 5.0 JDK 软件包。

(1) 单击“开始”按钮, 在弹出的菜单中依次选中“设置”/“控制面板”命令。

(2) 在“控制面板”窗口中双击“添加或删除程序”图标。

(3) 在弹出的“添加或删除程序”列表框中选中“J2SE Development Kit 5.0”选项, 如图 1-7 所示。单击“删除”按钮即可。

(4) 根据提示, 逐步完成 J2SE Development Kit 5.0 的卸载。

(5) 在“添加或删除程序”列表框中选中“J2SE Runtime Environment 5.0”, 单击“删除”按钮即可将其删除。

如果用户使用的是 Unix 系列的平台, 则卸载 JDK 的过程更为简单, 具体操作步骤如下。

(1) 找到 JDK 安装目录的 _uninst 子目录。

(2) 输入 ./uninstall.sh, 运行卸载程序。

(3) 根据提示, 逐步完成卸载工作。



图 1-7 卸载 JDK

1.1.3 NetBeans 的安装

在安装 NetBeans 5.0 之前, 首先要获取相应平台的安装软件包, 可以从 <http://www.netbeans.org/downloads/index.html> 下载所需的 NetBeans 5.0 安装软件包。NetBeans 在所有支持 Java 的操作系统平台上都可以运行。以下列出了可以使用 NetBeans 5.0 的平台。

- Microsoft Windows;
- Solaris 操作系统 (SPARC®和 X86 Platform Edition);
- Red Hat Linux;
- Mac OS X 10.3;
- Open VMS 7.2-1 或更高版本;
- 其他 Linux 发行版本;
- 其他 UNIX®平台, 如 HP-UX。

下面将以在 Windows 平台下安装与卸载 NetBeans 5.0 为例, 介绍安装 NetBeans 5.0 的方法。

(1) 进入存放 NetBeans 5.0 安装程序的目录, 双击 netbeans-5_0-windows.exe, 启动安装程序。

(2) 在经过短暂的初始化工作以后, 显示如图 1-8 所示的界面。

(3) 在如图 1-8 所示的界面中, 如果单击“Cancel”按钮则取消安装, 单击“Next”按钮继续安装。单击“Next”按钮后, 安装界面如图 1-9 所示。



图 1-8 安装 NetBeans 欢迎界面

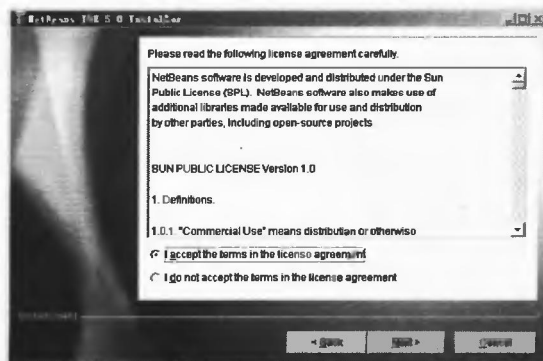


图 1-9 接受许可协议

(4) 选中 “I accept the terms in the license agreement” 单选按钮，单击 “Next” 按钮继续安装，安装界面如图 1-10 所示。

(5) 单击 “Browse” 按钮，选择安装目录，然后单击 “Next” 按钮，安装界面如图 1-11 所示。

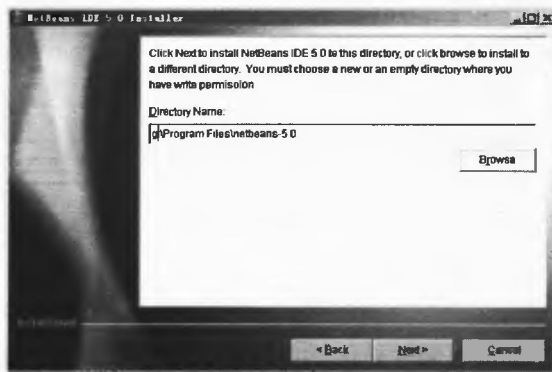


图 1-10 选择安装目录

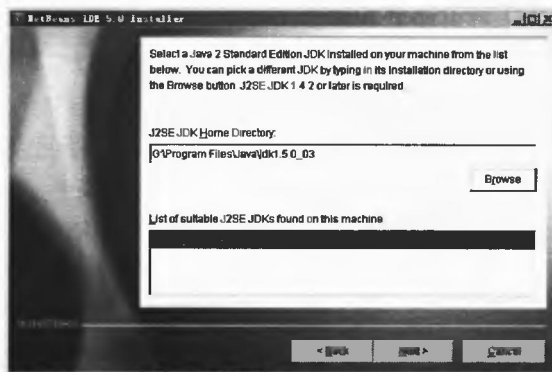


图 1-11 选择 JDK

(6) 单击 “Browse” 按钮，选择 Java SDK 的安装目录，选择完成以后，单击 “Next” 按钮。在一般情况下，NetBeans 的安装程序会自动寻找到已经安装的 JDK。

(7) 在弹出的 “信息提示” 窗口中，继续单击 “Next” 按钮，开始安装。等待一段时间以后，出现如图 1-12 所示界面。

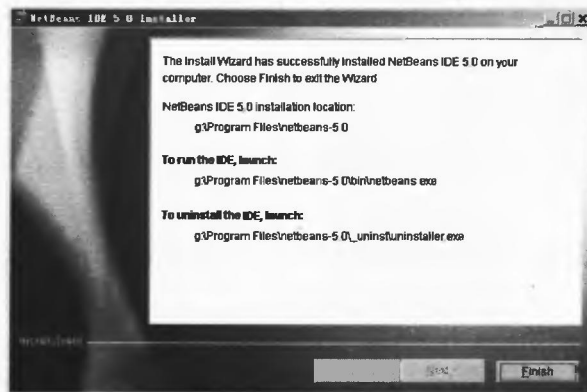


图 1-12 完成安装

(8) 在如图 1-12 所示的界面中, 单击“Finish”按钮完成安装。

安装完成后, 可以通过查阅日志文件 (%install directory%_uninst\install.log), 验证是否正确安装了该软件。如有必要, 可以随后删除下载的安装文件以恢复被占用的硬盘空间。

提示

在安装该程序之前, 必须确认已经拥有此安装程序的管理员权限。在安装过程中, 安装程序使用 %USERDIR%\Local Settings\Temp 目录保存临时文件。另外在其他操作系统 (Linux、Solaris) 中安装 NetBeans 5.0 的过程与在 Windows 下的安装过程基本相同, 读者可以参照 Windows 下的安装过程自行练习。

1.1.4 NetBeans 的卸载

下面介绍如何从 Windows 操作系统中卸载 NetBeans 5.0 软件包。

- (1) 单击“开始”按钮, 在弹出的菜单中依次选择“设置”/“控制面板”命令。
- (2) 在“控制面板”窗口中双击“添加或删除程序”图标。
- (3) 在弹出的“添加或删除程序”列表框中选“NetBeans 5.0 IDE”, 单击“删除”按钮。弹出如图 1-13 所示的窗口。
- (4) 单击“Next”按钮, 弹出图 1-14 所示的窗口。



图 1-13 卸载 NetBeans 界面

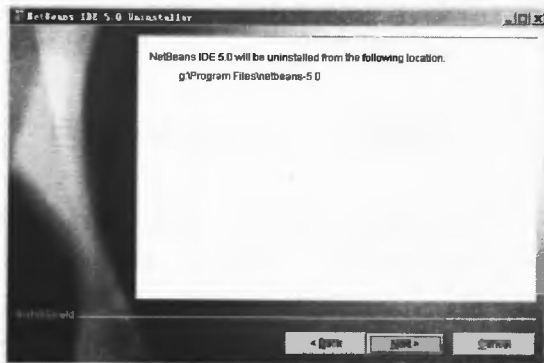


图 1-14 确认卸载 NetBeans

(5) 单击“Next”按钮开始卸载 NetBeans 5.0, 等待一段时间以后, 在弹出的窗口中单击“Finish”按钮完成卸载。

如果用户使用的是 UNIX 系列的平台, 则卸载 NetBeans 的操作, 步骤如下。

- (1) 找到 NetBeans 5.0 安装目录的 _uninst 子目录。
- (2) 输入 ./uninstall.sh, 运行卸载程序。
- (3) 根据提示, 逐步完成卸载工作。

1.2 NetBeans 的更新

NetBeans 能够自动从 NetBeans 更新中心下载并安装最新的更新包, 并对自身进行更新。为了使中国地区的用户更方便地使用这一功能, SUN 中国技术社区建立了一个 NetBeans 更新中心的本地镜像。用户可以按照如下的步骤修改 NetBeans 中的一些配置, 使得 NetBeans 可以从中国本地的更新中心进行自动更新。

- (1) 从菜单中选择“Tools”/“Options”选项，将弹出 IDE 的配置窗口。
- (2) 在配置窗口中单击“Advanced Options”按钮，弹出高级配置窗口，如图 1-15 所示。

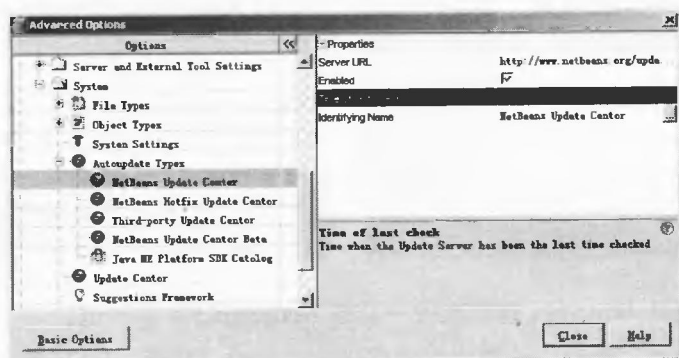


图 1-15 配置窗口

- (3) 展开窗口中 System 节点下的 Autoupdate Types 节点。
- (4) 对“Autoupdate Types”下的所有节点修改 Server URL 属性，将 Server URL 属性中的 `http://www.netbeans.org/` 修改为 `http://gceclub.sun.com.cn/NetBeans/`，注意保留 URL 的其他部分不变。
- (5) 配置完毕后，单击“Close”按钮退出配置窗口。

1.3 创建第一个 NetBeans 项目

经过前面几节的讲解，相信读者已经能够成功地安装和启动 NetBeans 5.0。本节将使用 NetBeans 5.0 开发一个简单的“Hello World”程序，以展示如何在 NetBeans 5.0 中创建、编译以及运行简单的项目。创建项目的步骤如下。

- (1) 选择 NetBeans 主菜单的“File”/“New Project”选项，将弹出“New Project”窗口，如图 1-16 所示。
- (2) 在如图 1-16 所示的界面中，分别选中“Categories”列表框中“General”选项和“Projects”列表框中“Java Application”选项，单击“Next”按钮，进入项目设置界面，如图 1-17 所示。

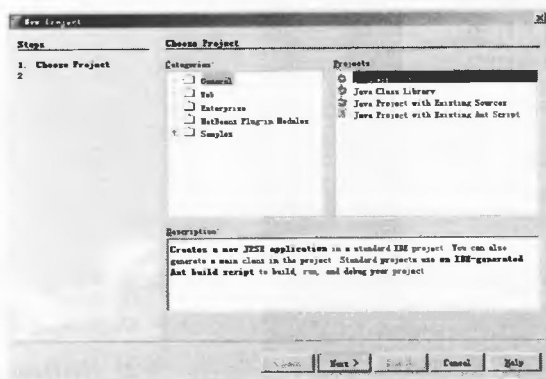


图 1-16 新建项目窗口

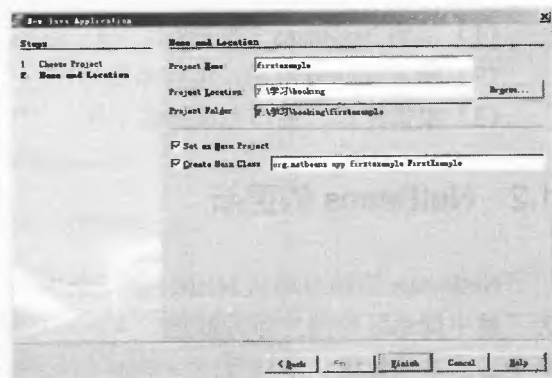


图 1-17 项目设置界面

(3) 在如图 1-17 所示的界面中, 指定项目的名称和位置。在此程序中, 将项目名称设置为 “firstexample”, 并通过单击 “Browse” 按钮来选择存储项目的目录。

提示 “Project Folder” 的内容不可编辑, 其值由 Project Name 和 Project Location 的值自动确定。

(4) 选中 “Set as Main Project” 和 “Create Main Class” 复选框。并将 “Create Main Class” 文本框的值修改为 `org.netbeans.app.firstexample.FirstExample`, 该值就是本项目主类的名称。

提示 在这里输入类的完整路径名称, NetBeans 会自动根据类名对应的包结构生成相应的目录结构。

(5) 单击 “Finish” 按钮完成项目的创建, 此时光标会自动出现在 `FirstExample.java` 的代码编辑窗口中, 如图 1-18 所示。

(6) 把光标移动到 `// TODO code application logic here` 处换行, 添加如下代码:

```
System.out.println("欢迎使用 NetBeans 5.0, 这是第一个例子");
```

(7) 按 “Ctrl+S” 键保存项目。

(8) 单击主菜单中 “Build” 菜单下的 “Build Main Project” 选项, 编译和打包应用程序。此时 “Output” 窗口在界面下方打开, 显示 Ant 脚本运行的结果, 如图 1-19 所示。



```
/*
 * FirstExample.java
 *
 * Created on 2006年3月7日, 上午1:56
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
package org.netbeans.app.firstexample;

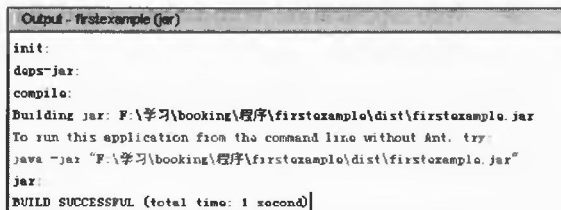
import java.io.IOException;

/**
 *
 * @author boyingking
 */
public class FirstExample {

    /** Creates a new instance of FirstExample */
    public FirstExample() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }
}
```

图 1-18 编辑器中代码



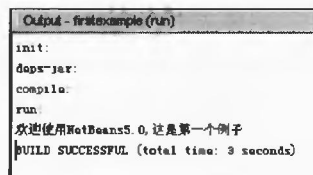
```
Output - firstexample.jar

init:
deps-jar:
compile:
Building jar: F:\学习\bookings\程序\firstexample\dist\firstexample.jar
To run this application from the command line without Ant, try:
java -jar "F:\学习\bookings\程序\firstexample\dist\firstexample.jar"
jar:
BUILD SUCCESSFUL (total time: 1 second)
```

图 1-19 编译输出内容

(9) 单击主菜单中 “Run” 菜单下的 “Run Main Project” 选项, 运行主项目。此时, 在 “Output” 窗口中会显示 Ant 脚本的运行结果, 以及程序所打印出的 “欢迎使用 NetBeans 5.0, 这是第一个例子” 消息, 如图 1-20 所示。

通过这个简单的例子, 读者对 NetBeans 有了初步的认识。



```
Output - firstexample (run)

init:
deps-jar:
compile:
run:
欢迎使用NetBeans5.0, 这是第一个例子
BUILD SUCCESSFUL (total time: 3 seconds)
```

图 1-20 运行输出内容

在下面的章节中将详细介绍如何使用 NetBeans。

1.4 NetBeans 中的项目模板

NetBeans 始终在项目内工作。IDE 项目由一组 Java 源文件和关联信息（如类路径中包含哪些内容、如何生成和运行项目等）组成。可以创建标准项目（使用 IDE 生成的 Ant 脚本来生成项目），也可以创建基于现有 Ant 脚本的自由格式项目。IDE 将项目信息存储在项目文件夹中，该文件夹包括 Ant 生成脚本、控制生成和运行设置的属性文件，以及将 Ant 目标映射到 IDE 的配置文件。

在 NetBeans 中包含很多不同种类项目的模版，开发人员在创建自己的项目时，并不一定要“从零做起”，也可以基于模版来创建项目，这样可以提高项目开发的效率。NetBeans 中主要包含以下标准项目模版。

- **Java Application**：用于创建带有主类的 Java SE 项目的模版。
- **Java Class Library**：用于创建不带主类的 Java 类库的模版。
- **Java Project With Existing Source**：基于已有的 Java 源代码创建 Java SE 项目的模版。
- **Web Application**：用于创建 Web 应用程序的模版。
- **Web Application With Existing Source**：基于已有的 Web 和 Java 源代码创建 Web 项目的模版。
- **Enterprise Application**：用于创建企业应用程序的模版。
- **Enterprise Application With Existing Source**：用于将已有企业应用程序导入标准 IDE 项目的模版。
- **EJB Module**：用于创建 Enterprise JavaBean 的模版。
- **EJB Module With Existing Source**：用于将 Enterprise JavaBean 模块导入标准 IDE 项目的模版。

NetBeans 还包含以下自由格式项目模版。

- **Java Project With Existing Ant Script**：基于已有的 Java 源代码创建 J2SE 项目的模版，并使用已有的 Ant 生成脚本来生成项目。
- **Web Application With Existing Ant Script**：基于已有的 Web 和 Java 源代码创建 Web 项目的模版，使用已有的 Ant 生成脚本来生成项目。
- **EJB Module With Existing Ant Script**：包含现有 Ant 脚本的 EJB 模块。用于将 EJB 模块导入 IDE 项目（使用用户自己的 Ant 生成脚本）的模版。

1.5 NetBeans 示例项目的使用

如果读者想尝试 NetBeans 的强大功能，但不想创建自己的项目，NetBeans 提供了丰富的示例项目给用户进行尝试。

创建示例项目时，NetBeans 会自动将示例代码复制到指定的目录下，同时还将自动生成必须的项目数据。创建示例项目的步骤如下。

- (1) 选择 NetBeans 主菜单中的“File”/“New Project...”选项。

(2) 在新建项目向导中选择“Samples”类别，在项目列表中选择任意的示例模版，如图 1-21 所示，单击“Next”按钮。

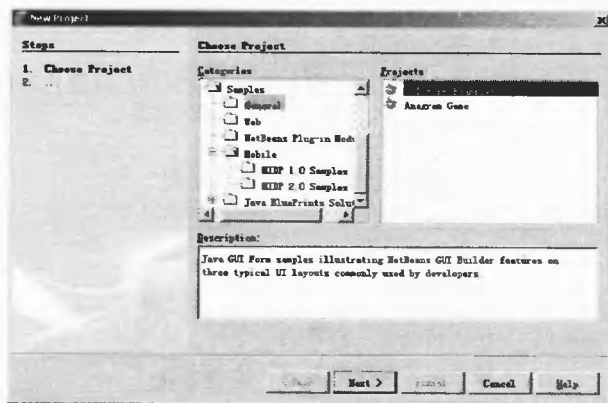


图 1-21 创建示例项目

(3) 在弹出的窗口中指定项目名称和目录，单击“Finish”按钮完成示例项目的创建。各个不同类别中包含的项目具体如下。

- General 中包含两个简单的 Java SE 应用程序项目。
- Web 中包含若干可以在 Tomcat 服务器上运行的示例项目。
- Mobile 中包含若干 Java ME 示例项目（这要求 NetBeans 安装了移动开发套件）。
- Netbeans Plug-in Modules 中包含一个简单的插件模块项目 FeedReader。
- Java BluePrints Solutions 中包含许多实用的 Java EE 应用程序设计模式模版项目，这些示例项目与 BluePrints 解决方案目录相对应，如果需要了解更多有关 BluePrints 信息的读者，可以到 SUN 的官方网站进行查找。

1.6 NetBeans 开发环境组件介绍

本节主要介绍如下内容。

- NetBeans 5.0 主菜单中各个菜单项的作用。
- NetBeans 5.0 中几个主要窗口的作用。

1.6.1 主菜单

菜单是任何一个 IDE 都不可或缺的组件。在 NetBeans 5.0 中一共提供了 12 个主菜单。每个主菜单的各个常用选项的功能的详细介绍如表 1-1～表 1-12 所示。

表 1-1 File 菜单常用选项功能

选 项	说 明
New Project	新建一个项目，NetBeans 5.0 包含了一些项目模版供用户选择
New File	新建一个文件，NetBeans 5.0 包含了一些文件模版供用户选择
Open Project	打开旧的项目
Open Recent Project	打开最近工作过的项目

续表

选 项	说 明
Close Project	关闭当前项目
Open File	打开一个已经存在的文件
Set Main Project	如果当前打开了多个项目，在此可以设置主项目
Project Properties	设置项目属性
Save	存储当前项目（使用当前文件名）
Save All	存储打开的所有项目（指定新文件名）
Refresh All Files	刷新所有文件
Page Setup	页面设置
Print	打印当前文件内容
Print To HTML	将当前文件保存为 HTML 文件
Exit	关闭当前项目并退出

表 1-2

Edit 菜单常用选项功能

选 项	说 明
Undo	取消前一次操作
Redo	重做被取消的操作
Cut	将当前选中的内容剪切到剪贴板
Copy	复制当前选中的内容到剪贴板
Past	将剪贴板中内容粘贴到指定位置
Delete	删除当前选中内容
Find Selection	搜索插入点所在的词出现的下一个位置
Find Next	查找搜索词出现下一个位置
Find Previous	查找搜索词出现上一个位置
Find	在当前选定的文件中搜索文本，源代码编辑器跳转到字符串出现的第一个位置并突出显示所有匹配的字符串
Replace	在当前选定的文件中替换文本
Find Usages	在 Usage 窗口中显示当前选定内容的具体位置
Find in Project	在当前项目中查找
Start Macro Recording	开始宏录制
Stop Macro Recording	结束宏录制

表 1-3

View 菜单常用选项功能

选 项	说 明
Editor	选择要显示的编辑器窗口，“Design”或“Source”窗口
Cold Folds	设置折叠与展开，包括全部折叠与展开或部分折叠与展开
Web Browser	打开系统默认的浏览器
Documentation Indices	文档索引
Toolbars	设置在工具栏中显示的内容，可以使用系统提供的内容，也可以自定义工具栏中的内容

续表

选 项	说 明
Show Line Numbers	显示行号
Show Editor Toolbars	显示编辑器工具栏
Show CVS State Label	显示 CVS 状态标签

表 1-4

Navigate 菜单常用选项功能

选 项	说 明
Go To Class	通过“Go To Class”对话框，可以快速打开文件，在该对话框中开始键入类名时，将显示所有与键入的前缀匹配的文件
Go To Previous Document	转到上一个工作文档，其内容在源代码编辑器窗口中显示
Go To Source	光标定位到“Source”窗口中
Go To Declaration	转到声明。与“Go To Class”类似，此快捷键将打开在插入点处声明的变量所在的文件
Back	转到当前选定文件的跳转列表的上一个位置。跳转列表是在编辑器中进行修改时所在的所有位置的历史记录
Forward	转到当前选定文件的跳转列表的下一个位置
Go To Line	光标定位到当前文件中指定的行
Toggle Bookmark	为光标所在的行增加一个书签或删除光标所在行已有的书签
Next Bookmark	光标定位到下一个书签所在行
Previous Bookmark	光标定位到上一个标签所在行
Next Error	光标定位到下一个错误的位置
Previous Error	光标定位到上一个错误的位置
Select in Projects	查找当前选定内容位于哪一个文件，并在 Projects 窗口中自动选中该文件
Select in Fields	查找当前选定内容位于哪一个文件，并在 Files 窗口中自动选中该文件
Select in Favorites	查找当前选定内容位于哪一个文件，并在 Favorites 窗口中自动选中该文件

表 1-5

Source 菜单常用选项功能

选 项	说 明
Overwrite Method	重写指定的方法
Surround With try-catch	如果没有选中内容，则将光标所在行的内容包含到 try-catch 块中。否则将选中区域内的所有内容都包含到 try-catch 块中
Reformat Code	重新指定代码格式
Shift Left	将光标所在行的内容向左挪动
Shift Right	将光标所在行的内容向右挪动
Comment	使用行注释标记 (“//”) 将当前行或所有选定行标记为注释
Uncomment	删除注释。此命令只适用于以行注释标记 (“//”) 开头的行
Insert Next Matching Word	正向查找匹配的单词并插入
Insert Previous Matching Word	反向查找匹配的单词并插入

表 1-6

Refactor 菜单常用选项功能

选 项	说 明
Rename	更改选定变量名称, 包括修改包名、方法名、字段名、类名
Extract Method	把指定的语句提取成方法
Change Method Parameters	更改选定方法的参数
Encapsulate Field	封装所选定的变量
Move Class	将当前选中类在项目内的 Source Packages 与 Test Packages 之间移动
Move Inner To Out Level	把内部类转换为普通类
Convert Anonymous Class To Inner	将匿名内部类转换为普通内部类
Extract Interface	让当前选中的类实现接口
Extract Superclass	让当前选中的类继承超类
Safely Delete	安全删除所选内容

表 1-7

Build 菜单常用选项功能

选 项	说 明
Build Main Project	编译主项目
Clean And Build Main Project	清除原有编译内容后编译主项目
Generate Javadoc For Project	为项目生成 Java 文档
Compile File	编译当前文件

表 1-8

Run 菜单常用选项功能

选 项	说 明
Run Main Project	运行主项目
Debug Main Project	调试主项目
Test Project	测试项目
Attach Debugger	配置连接调试器
Finish Debugger Session	完成调试器会话
Pause	暂停运行
Continue	继续运行
Step Over	越过, 用于代码的步进调试
Step Into	步入, 用于代码的步进调试
Step Out	步出, 用于代码的步进调试
Run To Cursor	运行到光标处, 光标以后程序不运行
Apply Change Codes	在运行中查看代码在编译后是否进行过修改, 如果已经修改, 则使用新内容作为运行时内容
Toggle Breakpoint	在设置的断点间进行切换
New Breakpoint	新建断点

表 1-9

CVS 常用菜单项及说明

选 项	功 能
Show All Changes	显示所有更改
Update All Changes	比较所有更改

续表

选 项	功 能
Commit All Changes	提交所有更改
Update Project with Dependencies	更新项目并保存依赖关系
Import into Repository	导入到库中
Export diff Patch	导出差异修补程序
Revert Modifications	还原修改

表 1-10

Tools 菜单常用选项功能

选 项	功 能
Javadoc Index Search	Java 索引搜索, 在弹出的对话框中可以输入要搜索的内容
Create JUnit Test	创建 JUnit 测试
Add To Palette	将当前所选中的类添加到 Palette 中
Auto Comment	为当前选中的类自动添加注释
Internationalization	配置国际化选项
Java Platform Manager	配置与管理 NetBeans 5.0 中已经安装的 Java ME Wireless Toolkit 和 JDK
Server Manager	配置与管理 NetBeans 5.0 中已经安装的 Java EE 服务器
NetBeans Platform Manager	配置与管理 NetBeans 5.0
Library Manager	配置与管理 NetBeans 5.0 的各种插件模块
Palette Manager	配置与管理 NetBeans 5.0 控件面板
Security Manager	安全管理器
Template Manager	配置与管理 NetBeans 5.0 的模版
Option	自定义设置选项
Module Manager	添加或删除模块
Update Center	更新中心, 可以在此更新模块

表 1-11

Window 菜单常用选项功能

选 项	功 能
Projects	显示 Projects 窗口
Files	显示 Files 窗口
Favorites	显示 Favorites 窗口
Output	显示 Output 窗口
Navigator	显示 Navigator 窗口
Palette	显示 Palette 窗口
Properties	显示 Properties 窗口
Runtime	显示 Runtime 窗口
Search Results	显示 Search Results 窗口
Find Usages Results	显示 Usages 窗口
JUnit Test Results	显示 JUnit Test Results 窗口, 此窗口以统计表和普通文本的形式输出测试结果
Refactoring Preview	显示 Refactoring Preview 窗口

续表

选 项	功 能
To Do	显示 To Do 窗口, 此窗口中包含已经打开文件的位置信息
HTTP Monitor	显示 HTTP Monitor 窗口, 此窗口中可以显示有关于 Request、Session、Cookies 等信息
Versioning	显示 Versioning 窗口
Inspector	显示 Inspector 窗口
Mobility Inspector	显示 Mobility Inspector 窗口
Debugging	调试信息输出项, 包括在 Local Variables、Watches、Call Stack、Classes、Breakpoints、Sessions、Threads、Sources 子窗口中输出
Close Window	关闭当前选中窗口
Maximize	如果当前窗口处于普通状态, 则最大化该窗口, 反之还原窗口
Close All Document	关闭所有在源代码编辑器和 GUI 设计器中打开的文档
Close Other Document	关闭除当前文档外的所有在源代码编辑器和 GUI 设计器中打开的文档
Document	查看已经打开文档的名称以及描述

表 1-12

Help 菜单常用选项功能

选 项	功 能
Help Content	显示 NetBeans 5.0 的帮助文档
Tutorials	几本 NetBeans 5.0 的使用手册, 包括 J2EE Tutorials In NetBeans IDE、NetBeans Platform Tutorials、Quick Start、NetBeans IDE
Java BluePrints Solutions Catalog	Java BluePrints 解决方案目录项目
Keyboard Shortcuts	显示 NetBeans 5.0 中定义的快捷键
J2ME MIDP Develop Quick Start Guide	显示 J2ME MIDP 快速开发使用手册

1.6.2 主环境

启动 NetBeans 后, 可以看到如图 1-22 所示的主环境。

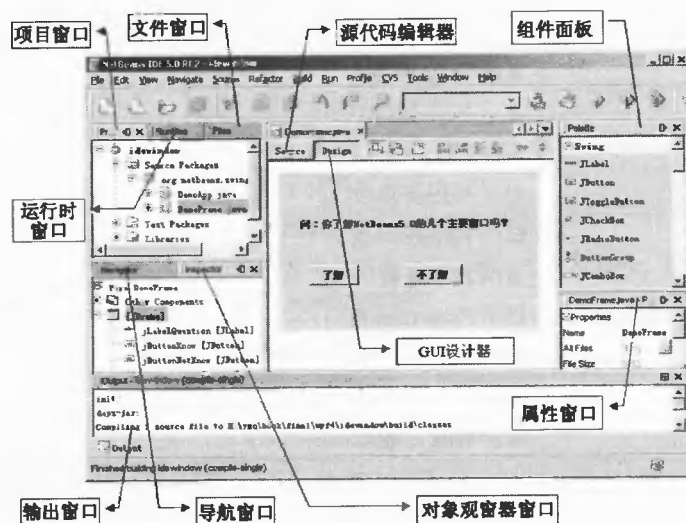


图 1-22 主环境及窗口

在图 1-22 中标出了主环境中几个常用的窗口。在接下来的几个小节中将会对每个窗口进行详细的介绍。

1.6.3 项目窗口

项目窗口是项目源的主入口点，显示重要项目内容（如 Java 包和 Web 页）的逻辑视图。选择主菜单中“Window”/“Projects”选项，打开项目窗口。项目窗口中的基本内容如图 1-23 所示。

可以看到，项目窗口中的内容是以树状形式组织和显示的。使用 Java Application 模版创建的项目，其根节点下主要包括 Source Packages、Test Packages、Libraries 和 Test Libraries 4 个节点。

1. Source Packages 节点

Source Packages 节点下的 `org.netbeans.swing.idewindow` 子节点，其值为项目的包名。展开 `org.netbeans.swing.idewindow` 节点，窗口内容如图 1-24 所示。

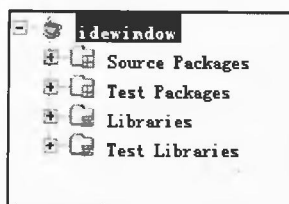


图 1-23 项目窗口

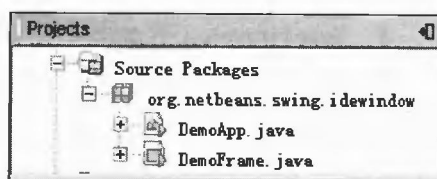


图 1-24 Source Packages 节点及子节点

可以看到，`org.netbeans.swing.idewindow` 包含 `DemoApp.java` 和 `DemoFrame.java` 两个子节点，表示该包中 Java 文件的文件名。展开 `DemoFrame.java` 节点，此时窗口内容如图 1-25 所示。

`DemoFrame.java` 节点包含 `DemoFrame` 和 `Form DemoFrame` 两个子节点，表示该 Java 文件中包含的类的名称，展开 `DemoFrame` 节点此时窗口内容如图 1-26 所示。从图中可以看到 `DemoFrame` 包含 `Fields`、`Constructors`、`Methods`、`BeanPatterns` 4 个子节点。

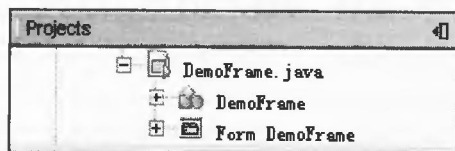


图 1-25 DemoFrame.java 节点及子节点

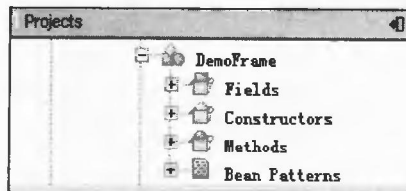


图 1-26 DemoFrame 节点及子节点

- Fields 的子节点表示该类的所有成员变量。
- Constructors 的子节点表示该类的所有构造器。
- Methods 的子节点表示该类中包含的方法。
- BeanPatterns 的子节点表示通过反射可以在该 bean 中使用的属性和其他模式。

2. Test Packages 节点

用来对项目进行测试的类都存放在 Test Packages 节点下，默认情况下，其子节点是由系统提供的 Default Package，开发人员可以在其中添加自己的测试类。展开 Test Packages 节点的情况如图 1-27 所示。

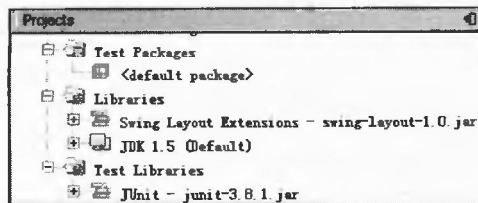


图 1-27 项目窗口中节点



注意

在本例中没有开发用于测试的类，所以 Test Packages 节点下没有具体的测试的类，关于测试的内容，请读者参照第 25 章的内容。

3. Libraries 节点

Libraries 节点包括项目中已经引入的包文件，根据用户所创建项目类型的不同，系统会自动帮助用户引入相应的包文件，开发人员也可以往其中添加新的要引入的包文件。展开 Libraries 节点的情况如图 1-27 所示。

4. Test Libraries 节点

Test Libraries 节点包括用来对项目进行测试的测试器，在默认情况下使用的是 JUnit 测试器，它的文件为 JUnit - junit-3.8.1.jar。展开 Test Libraries 节点的情况如图 1-27 所示。

对于以上列出的所有节点，都可以通过鼠标右键打开的快捷菜单来访问用于生成、运行、调试项目，以及打开“属性”对话框的上下文菜单。

1.6.4 文件窗口

文件窗口用于提供基于目录的物理视图，包括未在项目窗口中显示的文件和文件夹。从文件窗口中可以打开和编辑项目配置文件，如项目的生成脚本和属性文件，还可以查看项目生成的输出结果，如编译后的类文件、JAR 文件、WAR 文件和生成的 Javadoc 文档。在主菜单中选择“Window”/“File”命令，打开文件窗口，如图 1-28 所示。

在图中可以看到，根节点下有 build、dist、nbproject、src、test、build.xml 和 manifest.mf 7 个子节点。

- build 节点及其子节点用来表示编译后的类的存放路径以及编译后类文件的名称。如 DemoApp.class 和 DemoFrame.class 即为编译后的类名，而其父节点用来表示这两类的存放路径。完全展开 build 节点，如图 1-29 所示。
- dist 节点中内容表示项目生成的以及运行时需要用到的 jar 包。展开 dist 节点，如图 1-30 所示。lib 目录下的 swing-layout-1.0.jar 表示项目运行时需要用到的扩展包。idewindow.jar 节点则表示编译项目时生成的包。
- nbproject 节点中列出了该项目用到的所有的配置文件及属性文件。展开 nbproject 节点，如图 1-31 所示。
- src 节点用于表示源文件的存放路径以及源文件的名称，展开 src 节点，如图 1-32 所示。

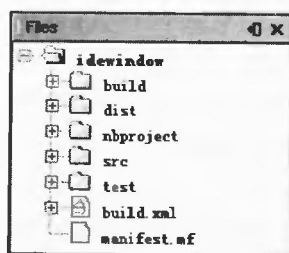


图 1-28 文件窗口

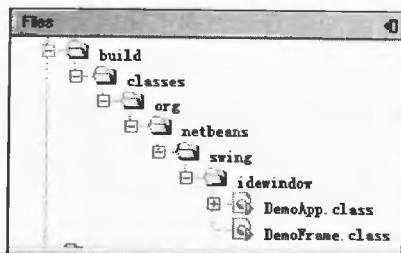


图 1-29 build 节点及子节点

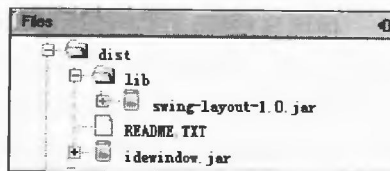


图 1-30 dist 节点及子节点



图 1-31 nbproject 节点及子节点

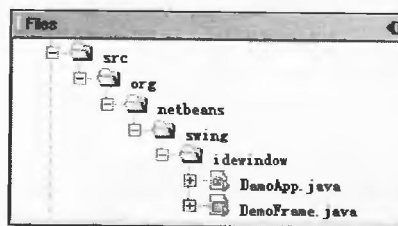


图 1-32 src 节点及子节点

- test 节点用于表示存放对项目进行测试的源文件的路径。
- build.xml 节点用于表示使用 Ant 构建项目时用到的配置文件。
- manifest.mf 节点用于表示构建项目时用到的清单文件。



提示 对于以上列出的所有节点，都可以通过用鼠标右键菜单来访问用于生成、运行、调试项目以及打开“属性”对话框的上下文菜单。

1.6.5 运行环境窗口

在运行环境窗口中，可以查看与配置开发时所要用的资源，如应用服务器连接、数据库连接等。单击“Window”/“Runtime”菜单，打开运行环境窗口，如图 1-33 所示。

- Server 节点中列出了已经添加的服务器，默认情况下安装 NetBeans 时，系统会自动安装 Tomcat 服务器。
- Processes 节点中显示了当前在 NetBeans 5.0 中运行项目的名称。如果当前没有项目处于运行状态，则显示“No Processes Running”。
- Databases 节点用来配置数据库的驱动程序以及创建数据库链接，默认情况下，系统提供了 JDBC-ODBC 桥驱动程序。
- 选中 HTTP Server 节点，单击鼠标右键，在弹出的快捷菜单中选择相应的选项，来启动或者关闭 HTTP 服务器。
- Web Services 节点用于添加和配置 Web 服务。
- DTD and XML Schema Catalog 节点中列出了可以使用的 DTD 和 XML 模版库。

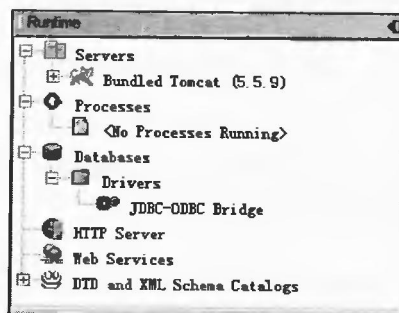


图 1-33 运行环境窗口

1.6.6 导航窗口

使用导航窗口，可以快速定位内容。选主菜单中“Window”/“Navigator”选项，打开导航窗口。默认情况下导航窗口处于 Members View 状态，窗口中显示的是，当前在项目窗口中被选中的 Java 文件或类的成员及构造器列表，如图 1-34 所示。

- 双击列表框中列出的构造器或方法，则光标会出现在代码编辑器中该构造器或方法签名所在的行。
- 双击列表框中的变量，则光标会出现在代码编辑器中该变量声明所在的行。

单击下三角按钮，选中下拉列表框中的“Inheritance View”选项，此时导航窗口处于 Inheritance View 状态，如图 1-35 所示。

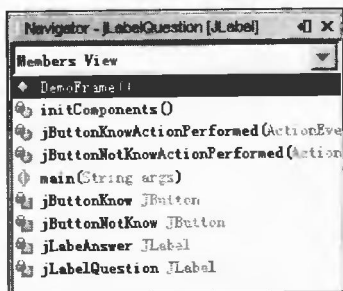


图 1-34 导航窗口

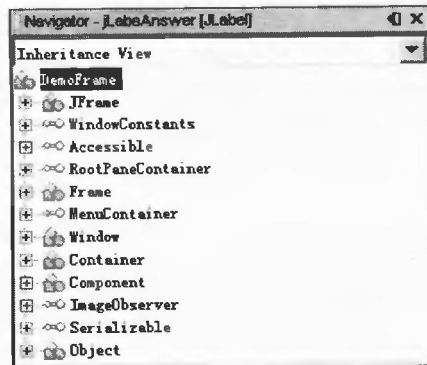


图 1-35 Inheritance View 状态导航窗口

在如图 1-35 所示的窗口中可以看出，选中“Inheritance View”选项之后，则会在列表框中以树状样式显示在 DemoFrame 类（DemoFrame 类在当前的项目窗口中被选中）中已经引用的系统类、接口，以及继承而来的方法。

1.6.7 对象观察器窗口

对象观察器窗口和导航窗口一样，也可以用来快速定位内容。选择主菜单中“Window”/“Inspector”选项，打开对象观察器窗口，如图 1-36 所示。

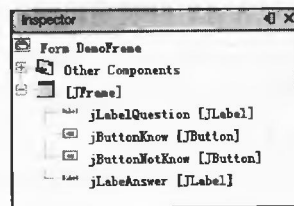


图 1-36 对象观察器窗口



提示

该窗口是用于列出当前被选中的顶层容器中各个可视化组件的逻辑包含关系。如果当前没有选中某个可视化界面组件，则对象观察器窗口不可用。

单击对象观察器窗口中的节点，则 GUI 设计器中相应的控件将处于被选中的状态。用鼠标右键单击该控件，选择快捷菜单中相应的选项，可以对控件进行各种操作，如修改控件的名称、为控件添加事件处理方法，以及设置控件的属性等。

1.6.8 源代码编辑器

源代码编辑器是用于编辑源代码的主要工具。尽管可视化编程技术大大减轻了程序员编写代码的工作量，但并不能完全取代原始代码的编写工作，代码的编写仍然是整个程序设计

的核心。一个程序的好坏在很大程度上取决于代码的编写。

源代码编辑器提供了可以使代码编写更简便、更快捷的各种功能，如代码完成、编译、错误突出显示、代码元素的语法突出显示，以及其他高级格式设置和搜索功能。

虽然可以将源代码编辑器视为单个 IDE 组件，但实际上是一个编辑器集合。每种类型的源文件都有可以提供不同功能的编辑器。在本小节中将主要介绍 Java 编辑器的使用方法。要在源代码编辑器中打开 Java 源文件，请在项目窗口或文件窗口中双击该文件的节点。代码编辑器如图 1-37 所示。



图 1-37 源代码编辑器






Java 源代码编辑器中具有代码自动完成功能，可以在键入几个字符后，从可能的类、方法和变量等内容列表中进行选择，然后自动完成表达式。源代码编辑器还包含一个 Javadoc 预览窗口，该窗口显示了当前代码完成框中所选内容的 Javadoc 文档（如果存在）。

源代码编辑器中提供了一些工具按钮，位于源代码编辑器的顶端，其功能如表 1-13 所示。

表 1-13 源代码编辑器中工具按钮的功能




图 标	作 用
	查找搜索词上一个出现的位置
	搜索插入点所在的词下一个出现的位置
	查找搜索词下一个出现的位置
	打开/关闭搜索结果突出显示
	为光标所在的行增加一个书签或删除光标所在行的已有书签
	光标定位到下一个书签所在的行
	光标定位到上一个书签所在的行
	转到当前选定文件跳转列表的下一个位置，跳转列表是在编辑器中进行修改时所有位置的历史记录
	转到当前选定文件跳转列表的上一个位置
	将当前行或选定内容向左移动一个制表符

续表

图 标	作 用
	将当前行或选定内容向右移动一个制表符
	启动宏录制
	关闭宏录制
	注释选中行
	取消注释选中行

1.6.9 GUI 设计器

如果希望使用 GUI 设计器，以可视方式编辑 Java GUI 窗体，则必须使用 Java GUI 窗体模版组中的模版创建窗体源文件。此模版组包含基于 AWT 和 Swing 窗体的模版。并且如果希望通过鼠标拖曳的方式向窗体中添加控件，则必须要使用该方式创建窗体。打开 GUI 设计器，如图 1-38 所示。

- 单击 GUI 设计器顶端的  按钮，可以对 GUI 设计器中的内容进行预览。需要注意的是，预览界面可能与运行界面不完全一致，请以运行界面为准。
- 单击 GUI 设计器顶端的  按钮与  按钮，可以在选择模式 (Selection Mode) 与连接模式 (Connection mode) 之间进行切换。

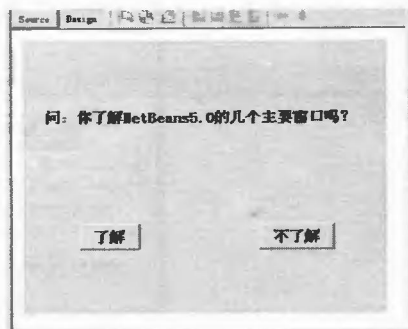


图 1-38 GUI 设计器

1.6.10 属性窗口

属性窗口也是开发项目过程中用得比较多的窗口，在其中可以设置选中条目的属性。通过选择主菜单中的“Window”/“Properties”命令，打开属性窗口，如图 1-39 所示。

在属性窗口中有 3 个按钮，分别为“Properties”、“Event”、“Code”。

- 在默认情况下，选中的是“Properties”按钮，此时可以通过属性窗口设置选中条目的基本属性，如控件的大小、位置、字体、颜色等，如图 1-39 所示。
- 单击“Event”按钮之后，属性窗口如图 1-40 所示。可以在该窗口中查看所选控件的事件处理方法，并可以为控件增加或删除事件处理方法。
- 单击“Code”按钮后，属性窗口如图 1-41 所示。可以在该窗口中查看与修改控件本身的一些信息，如控件名称、类型，以及访问限制修饰符等。

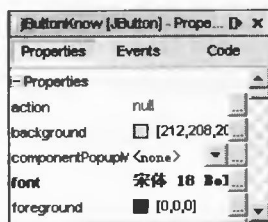


图 1-39 默认属性窗口

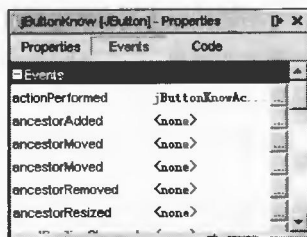


图 1-40 Event 状态属性窗口

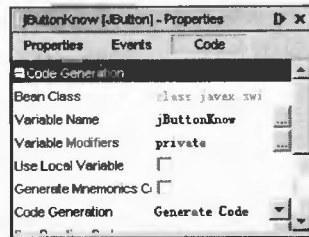


图 1-41 Code 状态属性窗口

1.7 NetBeans 常用的快捷键

在处理大型项目时，能否快速查找文件和字符串，在不同的窗口之间迅速切换，并迅速地编译、测试、运行项目，这些对于提高生产率至关重要。快捷键的使用将会使各种操作更加迅速、便捷。

1.7.1 基本常用快捷键

本节将以表格的形式，列出一些基本常用快捷键，如表 1-14~表 1-17 所示。

表 1-14 基本操作常用快捷键

快捷键	命令描述
Ctrl+Shift+N	新建一个项目
Ctrl+N	新建一个文件
Ctrl+Shift+O	打开一个项目
Ctrl+O	打开一个文件
Ctrl+S	保存当前文件内容
Ctrl+Z	取消上一次操作
Ctrl+Y	重做取消的操作
Ctrl+X	将当前选中的内容添加到剪贴板中
Ctrl+C	复制当前所选内容到剪贴板
Ctrl+V	将剪贴板中内容粘贴到当前位置
Delete	删除当前所选内容

表 1-15 编译与运行常用快捷键

快捷键	命令描述	快捷键	命令描述
F11	编译主项目	F5	调试主项目
Shift+F11	清除已编译内容后重新编译	Shift+F5	运行当前文件
F9	编译当前文件	Ctrl+Shift+F5	调试当前文件
F6	运行主项目	Shift+F1	在 Java 帮助文档中搜索

表 1-16 打开窗口常用快捷键

快捷键	命令描述	快捷键	命令描述
Ctrl+1	打开项目窗口	Ctrl+6	打开 To Do 窗口
Ctrl+2	打开文件窗口	Ctrl+7	打开导航窗口
Ctrl+3	打开收藏夹窗口	Ctrl+Shift+7	打开 Palette 窗口
Ctrl+4	打开输出窗口	Ctrl+Shift+8	打开属性窗口
Ctrl+5	打开运行时窗口		

表 1-17 源代码编辑器中常用快捷键

快捷键名称	作用
F3	查找搜索词下一个出现的位置

续表

快捷键名称	作用
Shift+F3	查找搜索词上一个出现的位置
Ctrl+F3	搜索插入点所在的词下一个出现的位置
Ctrl+F	在当前选定的文件中搜索文本, 源代码编辑器跳转到字符串出现的第一个位置并突出显示所有匹配的字符串
Ctrl+H	将找到的内容替换为指定内容
Alt+F7	在 Usage 窗口中查找
Alt+Shift+H	打开/关闭搜索结果突出显示
Ctrl+Shift+P	在项目中查找
Alt+L	转到当前选定文件跳转列表的下一个位置、跳转列表是在编辑器中进行修改时所有位置的历史记录
Alt+K	转到当前选定文件的跳转列表的上一个位置
Alt+Shift+L	转到所有文件(不是当前选定文件)中下一个跳转列表的位置
Alt+Shift+K	转到所有文件(不是当前选定文件)中上一个跳转列表的位置
Ctrl+Shift+F	重新设置整个文件或在源编辑器中选择的任何文本的格式
Ctrl+T	将当前行或选定内容向右移动一个制表符
Ctrl+D	将当前行或选定内容向左移动一个制表符
Ctrl+E	删除当前行
Ctrl+Shift+T	使用行注释标记("//")将当前行或所有选定行标记为注释
Ctrl+Shift+D	删除注释, 此命令只适用于以行注释标记("//")开头的行
Alt+Shift+R	重新命名选定内容

1.7.2 缩写展开快捷键

缩写展开快捷键即空格键。缩写展开是在源代码编辑器中使用缩写输入, 以减少键盘输入操作。输入缩写后按下空格键, 缩写会自动展开。例如, 如果输入 psfs 后并按空格键, 则其将展开为 public static final String。表 1-18 列出了 Java 源代码编辑器中提供的缩写字符及其对应展开后的字符。

表 1-18

缩写字符与展开后字符对照表

缩写字符	展开后字符	缩写字符	展开后字符
En	Enumeration	ca	catch (
Ex	Exception	cl	class
Ob	Object	cn	continue
Psf	public static final	df	default:
Psfb	public static final boolean	eq	equals
Psfi	public static final int	ex	extends
Psfs	public static final String	fa	false
St	String	fi	final
ab	abstract	fl	float
bo	boolean	fy	finally

续表

缩写字符	展开后字符	缩写字符	展开后字符
ie	interface	pst	printStackTrace();
im	implements	psvm	public static void main(String[] args){}
iof	instanceof	pu	public
ir	import	re	return
le	length	serr	System.err.println("");
newo	Object name = new Object(args);	sout	System.out.println("");
pe	protected	st	static
pr	private	sw	switch
psf	private static final	sy	synchronized
psfb	private static final boolean	th	throws
psfi	private static final int	trycatch	try {}catch (Exception e) {}
psfs	private static final String	tw	throw

提示 如果缩写与需要输入的文本相同（有时需要保持原样，不进行扩展），按“Shift”+“Spacebar”键可以防止字符展开。

1.7.3 代码自动修复快捷键

“Alt”+“Enter”组合键是源代码编辑器中非常重要的快捷键。当代码有错误时，该组合键可为开发人员提供一些可选的修复方案，从而提高代码调试的效率。如在源代码编辑器中某个类内部加入如下一段代码（在代码中并没有引入 java.awt 包）：

```
Button myButton=new Button();
```

在输入完成后，源代码编辑器中的代码如图 1-42 所示。此时按照如下步骤使用“Alt”+“Enter”组合键即可完成代码的自动修复。



（1）在该行的左侧出现了错误的提示图标 ，如果用鼠标单击出错的行会出现一个  图标，把鼠标放到该图标上，此时该行如图 1-43 所示。



图 1-42 输入代码后该行状态



图 1-43 鼠标移到灯泡上时该行状态

（2）根据提示按下“Alt”+“Enter”组合键，该行如图 1-44 所示。

（3）可以看到，按下“Alt”+“Enter”组合键后，出现了提示信息“Add import for java.awt.Button”，说明需要用到的 java.awt.Button 类没有被导入。

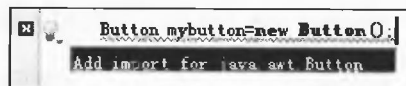


图 1-44 按下“Alt”+“Enter”后该行状态

（4）单击“Add import for java.awt.Button”后，系统将自动完成导入工作。此时查看源代码，可以看到 java.awt.Button 包已经被成功导入。

（5）在成功导入 java.awt.Button 包以后，可以看到错误提示消失，代码修复完成。

1.8 自定义 NetBeans 开发环境

NetBeans 5.0 是一种功能非常强大的集成开发环境，用户可以使用系统默认设置下的开发环境，也可以对开发环境进行很多自定义设置。本节将介绍自定义 NetBeans 5.0 的开发环境。

选择主菜单中“Tools”/“Options”选项，在弹出的窗口中单击左下角的“Advanced Options”按钮，出现高级自定义设置窗口，如图 1-45 所示。

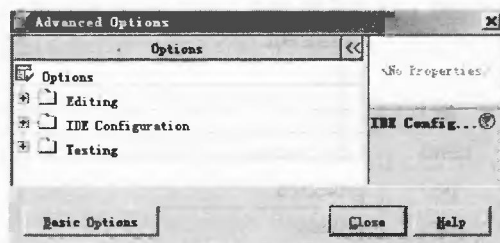


图 1-45 高级设置窗口

从图中可以看到，根节点下有 3 个子节点，分别为 Editing、IDE Configuration 和 Testing，3 个节点的主要作用如表 1-19 所示。

表 1-19 3 个节点的说明

节点名称	说 明
Editing	与编辑有关的设置项目在此节点中
IDE Configuration	与 IDE 配置有关的设置项目在此节点中
Testing	与 JUnit 测试有关的设置项目在此节点中

展开 Editing 节点，如图 1-46 所示。

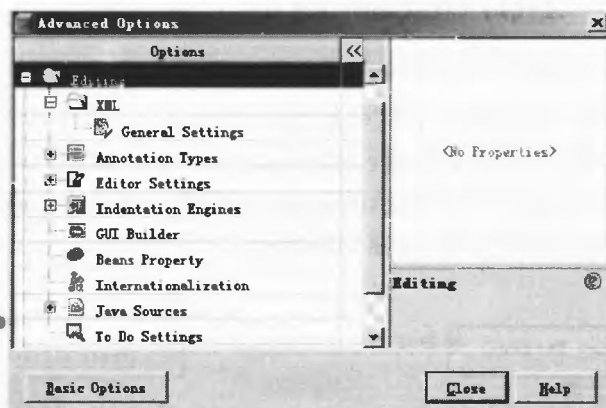


图 1-46 Editing 节点及其子节点

Editing 中各个主要子节点的功能如表 1-20 所示。

表 1-20 “Editing”中主要子节点的功能说明

节点名称	功能说明
General Settings	设置 XML
Annotation Types	设置各种提示或显示信息的格式，如颜色、亮度、字体大小
Editor Settings	设置 NetBeans 中的各种编辑器的属性
Indentation Engines	设置各种编辑器中的缩进方式

续表

节点名称	功能说明
GUI Builder	设置 GUI Builder 的各种属性
Beans Property	设置 bean 的属性
Internationalization	设置国际化选项

IDE Configuration 节点主要用来进行一些与 IDE 有关的设置, 展开 “IDE Configuration” 节点, 如图 1-47 所示。

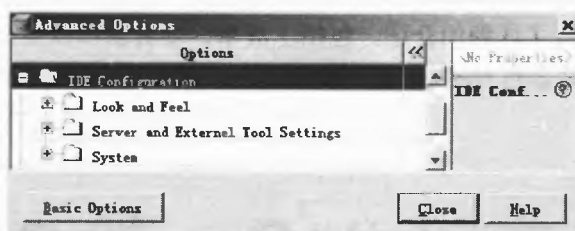


图 1-47 IDE Configuration 节点及其子节点

可以看到, 在 IDE Configuration 节点下面有 3 个子节点, 分别为 Look and Feel、Server and External Tool Settings 和 System。各个节点主要功能如表 1-21 所示。

表 1-21 “IDE Configuration” 中主要子节点的功能说明

节点名称	功能说明
Look and Feel	设置动作按钮、菜单、工具按钮的外观
Server and External Tool Settings	对服务器与一些扩展工具进行配置, 如配置使用的浏览器, 配置数据库与 J2EE 服务器
System	系统设置, 如设置系统的文件类型、对象类型更新中心和打印设置

Testing 节点中只包括 JUnit Module Setting 一个子节点, 可以用来对 JUnit 测试模块的各个选项进行设置, 如图 1-48 所示。

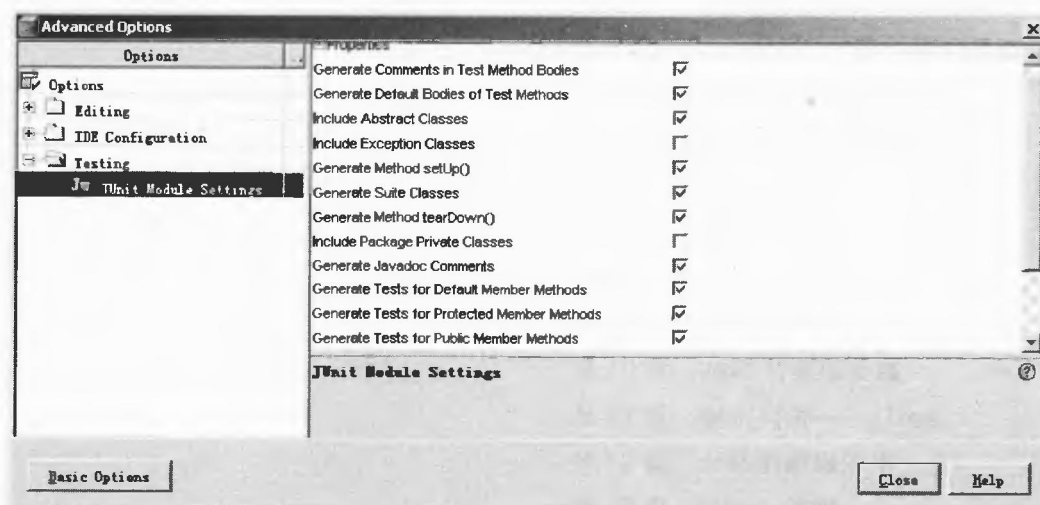


图 1-48 设置 JUnit 属性

在自定义环境的过程中,如果想要对某个选项进行设置,只需在窗口中选中相应的节点,然后就可以在窗口右边的列表中进行设置。

NetBeans 5.0 提供了非常强大的自定义开发环境的功能,本节仅对自定义开发环境做一个简要的说明,详细的设置情况,请读者参阅相关的使用手册或在线文档。

1.9 小结

本章介绍了如何搭建 NetBeans 开发环境,如何使用 NetBeans 的示例项目,并对 NetBeans 开发环境的各个组件进行了详细的介绍,包括主菜单、项目窗口以及文件窗口等内容。通过本章的学习,读者可以体会到 NetBeans 提供的是一套非常人性化的集成开发环境,非常有利于学习和使用。

当前使用 Java 开发具有图形用户界面的程序时，很多时候都会用到 Swing/JFC 控件。
本篇为 Swing/JFC 篇，主要介绍了如下内容。

- 什么是 Swing;
- Swing 的事件模型;
- Swing 控件类;
- Swing 容器类;
- Swing 的布局管理器;
- Swing 对话框;
- Swing 菜单;
- 图形与图像;
- 树状结构;
- Swing 表格;
- Swing 的线程。

第 2 篇 GUI 程序设计： Swing/JFC 篇

- 第 2 章 Swing 概述
- 第 3 章 Swing/JFC 的事件模型基础
- 第 4 章 Swing 常用基本控件
- 第 5 章 Swing 常用高级控件
- 第 6 章 Swing 容器类
- 第 7 章 布局管理器
- 第 8 章 Swing 对话框
- 第 9 章 Swing 菜单
- 第 10 章 Java 与图像处理
- 第 11 章 树状列表——JTree
- 第 12 章 表格的高级应用
- 第 13 章 Swing 线程

第 2 章

Swing 概述

在第 1 章对 NetBeans 5.0 的集成开发环境的各个组成部分做了详细的讲解。从本章开始, 将为读者介绍 NetBeans 中的 Swing 编程。本章首先对 Swing 做了一个简单的说明, 接着介绍了 Swing 的一些特性, 并将其与 AWT 进行了一些比较。最后, 以一个示例的形式介绍了在 NetBeans 中通过 Swing 进行简单的界面编程。

2.1 初识 Swing

2.1.1 什么是 Swing

Swing 是 SUN Microsystems 建立的新一代 GUI 工具包, 允许使用 Java 语言进行企业级开发。通过 Swing 可以建立包含许多控件的 Java 客户端应用程序, 并且很容易地扩充或者修改这些控件以控制其外观。

尽管 Swing 的开发独立于核心 Java 开发工具包, 但其要求 JDK 版本至少为 JDK 1.1.5。Swing 建立于事件模型之上, 而从 JDK 1.1 开始引入了这些模型, 因此 JDK 1.1 之前的版本不能使用 Swing 库。JDK 1.2 以后, Swing 已经成为 Java 标准包的一部分。

Swing 库中包含很多的程序包, 在表 2-1 中列出了 Swing 库中的包及其功能的描述。

表 2-1 Swing 中各个程序包功能简述

包 名	功 能 描 述
javax.accessibility	包含可以用来允许辅助技术与 Swing 控件进行交互的类和接口
javax.swing	包含核心 Swing 控件, 其中包含绝大多数模型接口和支持类
javax.swing.border	包含对抽象边框的定义和 8 个预定义的边框
javax.swing.colorchooser	包含对 JColorChooser 控件的支持
javax.swing.event	定义了几个 Swing 控件, 用于在类之间传达异步信息的新的监听器和事件
javax.swing.filechooser	包含对 JFileChooser 控件的支持
javax.swing.plaf	定义了为每个 Swing 控件建立可插拔外观风格的惟一元素。其各种子程序包在平台到平台的基础上为每个控件绘制单独的外观风格

续表

包 名	功 能 描 述
javax.swing.table	为表格控件提供模型和视图，表控件允许以网格形式，使用类似于电子表格的外观来排列各种信息
javax.swing.text	提供基于文本类的接口
javax.swing.text.html	对通过辅助编辑器工具包阅读和格式化 HTML 文本有很大作用
javax.swing.text.html.parser	包含对解析 HTML 的支持
javax.swing.text.rtf	对通过辅助编辑器工具包阅读和格式化丰富文本格式有很大作用
javax.swing.tree	定义了用于分层的树
javax.swing.undo	包含了用来实现撤销功能的类

提示 在 Swing 库的所有程序包中，使用最为广泛的程序包是 javax.swing。除了用于树、表格和文本控件的边框以及支持类外，几乎所有的 Swing 控件都被包含在该程序包中。

2.1.2 一个简单的 Swing 程序

本节将介绍一个使用 Swing 开发的例子，以介绍 Swing 中容器和组件的应用。这里的界面包括一个 JFrame 窗口、两个 JLabel、一个 JTextField 文本框、一个 JPasswordField 密码框和两个 JButton 按钮。程序运行界面如图 2-1 所示。

按照如下步骤来设计此程序。

(1) 界面中的所有组件都是 Swing 组件，所以要导入 javax.swing 包；容器采用了 AWT 中定义的布局管理器，所以还要导入 java.awt 包。

(2) 建立一个扩展 JFrame 的类 FirstSwing，用来存放各个组件。

(3) 在 main() 方法中新建一个 FirstSwing 的实例 mySwing。

(4) 在 FirstSwing 类中声明各个组件，并将其访问修饰符设为 private，代码如下：

```
private JLabel jLabelUserName;
private JLabel jLabelPassword;
private JTextField jTextFieldUserName;
private JPasswordField jpfPassword;
private JButton jButtonEnter;
private JButton jButtonCancel;
```

(5) 将 FirstSwing 的布局管理器设置为 null，代码如下：

```
this.setLayout(null);
```

(6) 为声明的每个组件创建实际对象，代码如下：

```
jLabelUserName=new JLabel("用户名:");
jLabelPassword=new JLabel("密码:");
jTextFieldUserName=new JTextField();
jpfPassword=new JPasswordField();
```

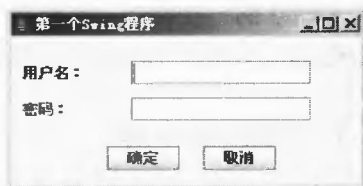


图 2-1 程序运行界面


```
jButtonEnter= new JButton("确定");  
jButtonCancel=new JButton("取消");
```

(7) 设置控件的位置, 代码如下:

```
jLabelUserName.setBounds(10,20,80,20);  
jLabelPassword.setBounds(10,50,80,20);  
jTextFieldUserName.setBounds(100,20,150,20);  
jpfPassword.setBounds(100,50,150,20);  
jButtonEnter.setBounds(80,90,60,20);  
jButtonCancel.setBounds(160,90,60,20);
```

(8) 将控件添加到容器中, 代码如下:

```
this.add(jLabelUserName);  
this.add(jLabelPassword);  
this.add(jTextFieldUserName);  
this.add(jpfPassword);  
this.add(jButtonEnter);  
this.add(jButtonCancel);
```

(9) 设置 FirstSwing 的大小、标题和可视性, 代码如下:

```
this.setBounds(330,250,300,150);  
this.setTitle("第一个 Swing 程序");  
this.setVisible(true);
```

这样, 整个程序的设计就完成了, 下面给出该示例的完整源代码 (FirstSwing.java):

```
import java.awt.*;  
import javax.swing.*;  
public class FirstSwing extends JFrame {  
    private JLabel jLabelUserName;  
    private JLabel jLabelPassword;  
    private JTextField jTextFieldUserName;  
    private JPasswordField jpfPassword;  
    private JButton jButtonEnter;  
    private JButton jButtonCancel;  
    public FirstSwing() {  
        this.setLayout(null); //设置布局管理器为 null  
        //初始化各个控件  
        jLabelUserName=new JLabel("用户名:");  
        jLabelPassword=new JLabel("密码:");  
        jTextFieldUserName=new JTextField();  
        jpfPassword=new JPasswordField();  
        jButtonEnter= new JButton("确定");  
        jButtonCancel=new JButton("取消");  
        //设置控件位置  
        jLabelUserName.setBounds(10,20,80,20);  
        jLabelPassword.setBounds(10,50,80,20);
```

```
jTextFieldUserName.setBounds(100,20,150,20);
jpfPassword.setBounds(100,50,150,20);
jButtonEnter.setBounds(80,90,60,20);
jButtonCancel.setBounds(160,90,60,20);
//将控件添加到面板上
this.add(jLabelUserName);
this.add(jLabelPassword);
this.add(jTextFieldUserName);
this.add(jpfPassword);
this.add(jButtonEnter);
this.add(jButtonCancel);
//设置窗体的大小、位置、可见性以及窗体的标题
this.setBounds(330,250,300,150);
this.setTitle("第一个 Swing 程序");
this.setVisible(true);
}

public static void main(String args[]){
    FirstSwing mySwing=new FirstSwing();
}
}
```

2.2 Swing 的特性

2.2.1 轻量级控件

大多数 Swing 控件都是轻量级控件，即这些控件并不依赖于本地对等控件。轻量级控件的功能首先在 JDK 1.1 出现。

对于轻型控件，每个控件都使用 Graphics 对象的绘图基本元素来绘制。在 JDK 1.1 以后，程序员可以直接扩展 java.awt.Component 或者 java.awt.Container 类，这些类不依赖于本地对等对象，并且允许开发人员快速建立容器的图形环境。

2.2.2 多种外观风格

目前外观风格已经成为 GUI 开发中的一个重要的内容。Swing 能够模拟很多种外观风格，目前支持 Windows、UNIX Motif、Linux GTK、Mac OS X 和本地 Java Metal 外观风格。

Swing 允许用户在运行时切换外观风格，而不必重启应用程序。这样，用户可以在程序运行时通过即时反馈来查看哪一个外观风格对自己是最好的。如果用户需要，开发人员还可以建立自己的外观风格，为自定义的外观风格指定行为模式和特定的外观。

2.2.3 更多的附加特性

Swing 中还提供了一些不同于 AWT 附加特性。

- Swing 中有许多新的控件，比如表格、树、滑杆、微调控制、进度条、内部框架和文本控件等。
- Swing 控件支持包括任意数量的嵌套边框的镶边替代物。

- Swing 控件有高效的工具提示。工具提示是一个弹出式文本，当鼠标指针停留在控件的绘制区域时，这个提示会立即出现。工具提示可以提供有疑问控件的更多信息。
- 可以将键盘事件绑定到控件中，规定其在给定条件下对各种按键做出反应。

2.3 Swing 与 AWT

AWT 是 Abstract Window ToolKit（抽象窗体工具包）的缩写，该工具包不仅为应用程序用户界面的设计提供了基本的组件，还提供了丰富的事件处理接口。

Swing 并不是 AWT 的替代品，而是在 AWT 基础上构建的一套新的图形界面系统。其提供了 AWT 所能提供的所有功能，并且用纯粹的 Java 代码对 AWT 的功能进行了大幅度的扩充，使得 Swing 具有更丰富并且更方便的用户界面元素集合，同时也使得对底层平台的依赖更少。

2.4 使用 NetBeans 开发 Swing 应用程序

NetBeans 为开发 Swing 程序提供了强大的支持。在 NetBeans 中，所有的 Swing 控件都可以通过鼠标拖放的方式添加到容器中，使得程序设计人员可以从繁重的界面设计中解放出来，把更多的时间放在实现程序的业务逻辑上。

下面将开发一个“添加学生信息”的用户界面，介绍如何在 NetBeans 中开发简单的 Swing 程序，程序运行界面如图 2-2 所示。启动 NetBeans 5.0，按照如下步骤来完成该界面的设计。

（1）选择“File”/“New Project”选项。出现如图 2-3 所示的窗口。

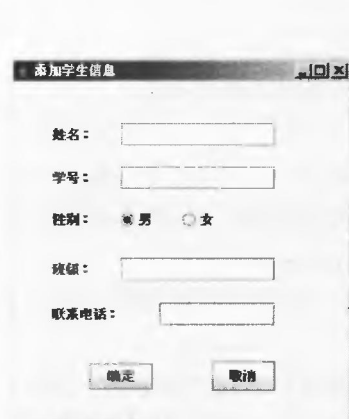


图 2-2 程序运行界面

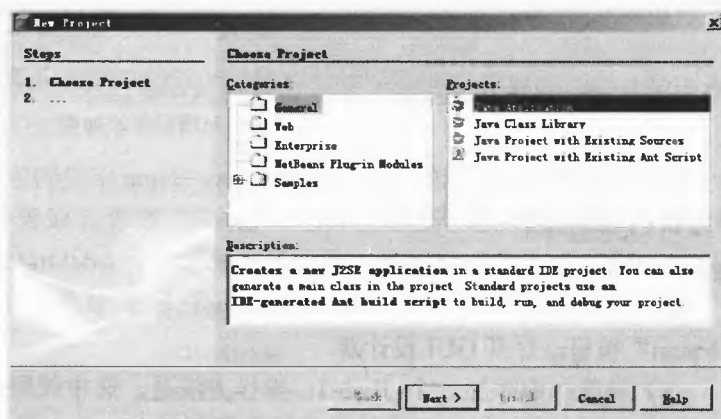


图 2-3 选择项目类型窗口

（2）可以在如图 2-3 所示的窗口中选择要创建的项目类型。在此程序的 Categories 列表框中选中“General”选项，在 Projects 列表框中选中“Java Application”选项，单击“Next”按钮，界面如图 2-4 所示。

（3）将 Project Name 的值修改为 addstudentinfo，该值为要创建的项目的名称。将 Create Main Class 的值修改为 org.netbeans.swing.addstudentinfo.AddStudent，该值为项目主类的名称。单击“Finish”按钮完成设置。此时已创建了基于 Java 应用程序的项目。

(4) 此时, 项目窗口中内容如图 2-5 所示。

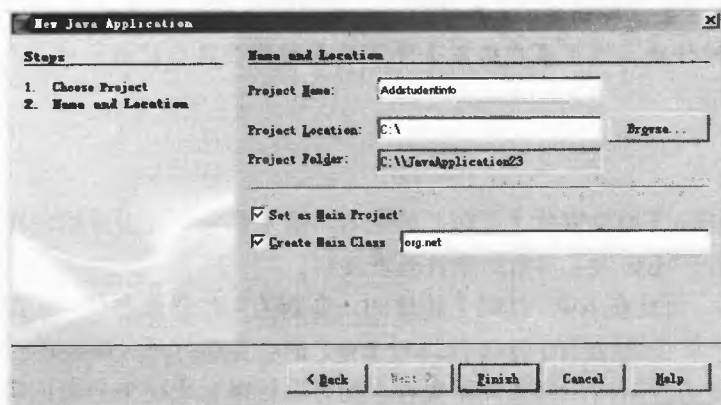


图 2-4 创建项目窗口

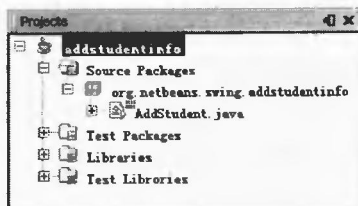


图 2-5 项目窗口中内容

(5) 选中 `org.netbeans.swing.addstudentinfo/AddStudent.java` 节点, 单击鼠标右键, 依次选中 “New” / “JFrame Form” 选项, 弹出如图 2-6 所示的窗口。

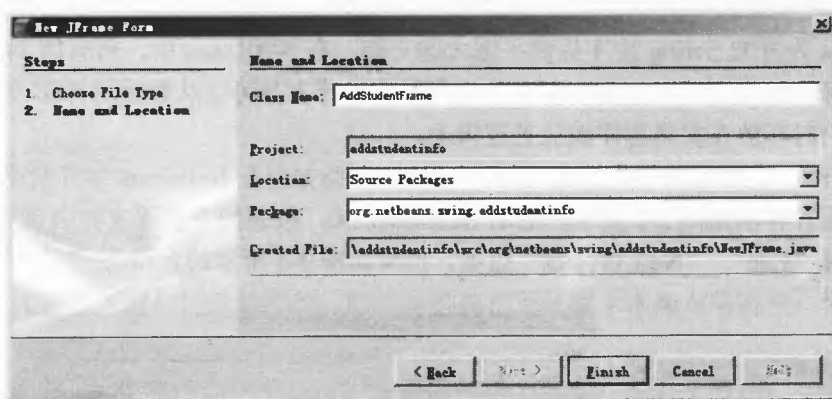


图 2-6 向项目中添加类

(6) 在如图 2-6 所示的窗口中, 将 Class Name 字段的值修改为 `AddStudentFrame`, 该值即为新创建的窗体类的类名, 单击 “Finish” 按钮完成窗体类的创建。此时项目窗口中 `org.netbeans.swing.addstudentinfo` 节点下会增加一个 `AddStudentFrame.java` 子节点。

(7) 在项目窗口中选中 `AddStudentFrame.java` 节点, 在对应的代码编辑器窗口中按下 “Design” 按钮, 打开 GUI 设计器。

(8) 向窗体中添加一个 `JLabel`。操作方法是, 选中代码编辑窗口右边 “palette” 面板上的 “JLabel” 选项, 在 GUI 设计器中的容器面板上, 单击鼠标左键, 此时即可向窗体中添加一个 `JLabel` 控件。

提示

可以用鼠标左键拖动控件以调整该控件的大小和位置。如果 “palette” 面板没有打开, 则选择主菜单中的 “Window” / “palette” 选项即可。

(9) 在 GUI 设计器中选中步骤 (8) 中添加的 `JLabel` 控件, 单击鼠标右键, 在弹出的菜单中选择 “Change Variable Name...” 选项, 在弹出的对话框中将标签的名称修改为 `jLabelStudentName`, 单击 “OK” 按钮完成修改, 如图 2-7 所示。

(10) 选中 `jLabelStudentName`，单击鼠标右键，在弹出的菜单中选中“Properties”选项，弹出如图 2-8 所示的对话框。

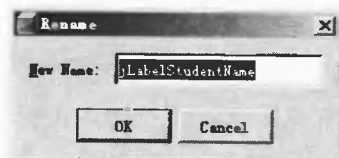


图 2-7 修改 JLabel 的名称

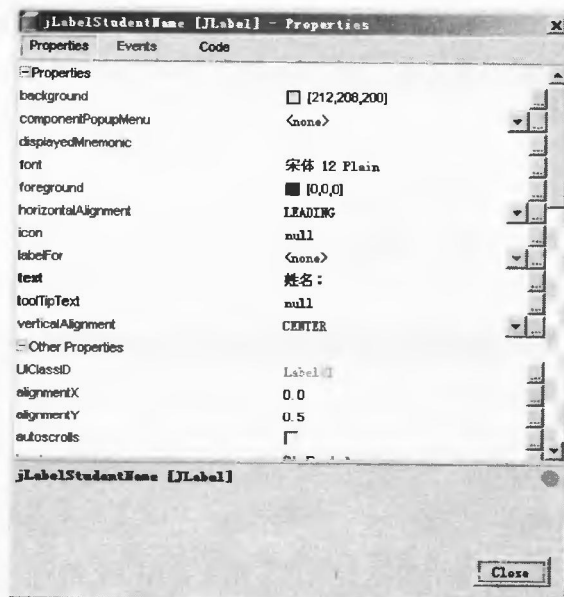


图 2-8 设置 JLabel 的属性

提示 在如图 2-8 所示的属性对话框中可以修改选中控件的各个属性设置。

(11) 在如图 2-8 所示对话框中修改 `jLabelStudentName` 的属性，将 `text` 属性的值修改为“姓名:”，单击“Close”按钮完成设置。

(12) 依次向容器中添加表 2-1 中所列的控件，通过鼠标的拖放设置好各个控件的大小和位置，并按表 2-2 修改控件的名称和 `text` 属性。

表 2-2 添加控件的名称和 `text` 属性

控 件	名 称	text 属性值
JLabel	<code>jLabelStudentID</code>	学号:
JLabel	<code>jLabelGender</code>	性别:
JLabel	<code>jLabelGrade</code>	班级:
JLabel	<code>jLabelPhoneNum</code>	联系电话:
JTextField	<code>jTextFieldStudentID</code>	空
JTextField	<code>jTextFieldGrade</code>	空
JTextField	<code>jTextFieldPhoneNum</code>	空
JRadioButton	<code>JRadioButtonMale</code>	男
JRadioButton	<code>JRadioButtonFemale</code>	女
JButton	<code>JButtonEnter</code>	确定
JButton	<code>JButtonCancel</code>	取消
JButtonGroup	<code>JButtonGroupGender</code>	

(13) 将 `JRadioButtonMale` 的 `selected` 复选框选中, 并将 `JRadioButtonMale` 与 `JRadioButtonFemale` 的 `buttonGroup` 属性设置为 `JButtonGroupGender`。

Swing 中的控件分为可视控件与非可视控件两种, 可视控件可以通过修改其属性来改变外观。而前面添加的 `JButtonGroupGender` 属于非可视控件, 非可视控件在设计时见不到其外观, 仅在对象观察器窗口中使用一个由其名称标识的节点来表示。选中该节点, 单击鼠标右键, 在弹出的菜单中选择“Properties”选项之后, 在属性窗口中设置其属性。

(14) 将 `JFrame` 的 `Title` 属性的值设置为“添加学生信息”。

(15) 将下列代码添加到 `AddStudent.java` 源代码编辑器中的 `// TODO code application logic here` 处。

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new AddStudentFrame().setVisible(true);  
    }  
});
```

(16) 编译并运行项目, 结果如图 2-2 所示。

在本节中介绍了一个非常简单的 Swing 程序界面的开发过程, 读者可以从中发现, 使用 NetBeans 进行 Swing 程序界面的可视化开发非常方便。

2.5 小结

本章对什么是 Swing 以及 Swing 的特性进行了简单的介绍, 并通过 Swing 与 AWT 的对比, 帮助读者加深对 Swing 的诸多特性的理解。在本章中, 还在 NetBeans 中开发了第一个 Swing 项目。在以后的章节中, 将会对如何在 NetBeans 中开发 Swing 程序进行详细的介绍。

第 3 章

Swing/JFC 的事件模型基础

对于图形用户界面的编程来说,事件处理是非常重要的。只有掌握了事件编程,才能熟练地使用 Java 进行图形用户界面开发。本章从事件处理的基础开始讲起,接着给出了 AWT 的事件层次图,然后分别介绍了动作事件、键盘事件、鼠标事件和窗口事件,最后讲解了事件适配器。

3.1 概述

任何支持 GUI 的操作环境都会不断地监听事件。当按下键盘或者单击鼠标时,便会触发一个相应的事件,操作环境把这些事件报告给正在运行的程序,接着程序来决定如何响应这些事件。在 Java 中,开发人员能够控制各种事件,可以把迅速响应事件的对象指派成一个事件监听器。

在 Java 中,关于事件的信息被封装在一个事件对象中。所有的事件对象都是从 `java.util.EventObject` 继承而来,而对于不同的事件类型,又都有其具体的子类,如 `KeyEvent` 和 `MouseEvent`。事件处理过程如图 3-1 所示。

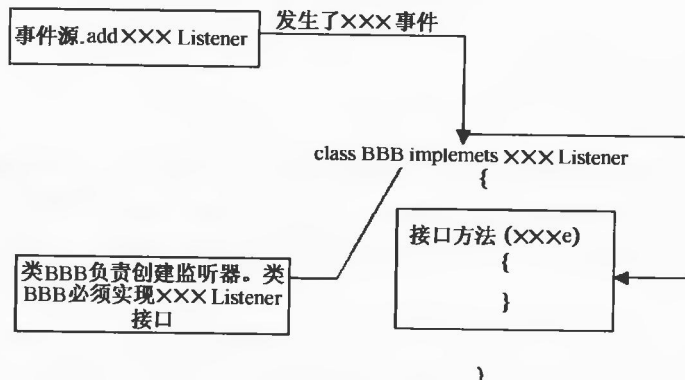


图 3-1 事件处理过程示意图

下面是 Java 中事件处理机制的描述。

- 监听器对象是一个实现了专门的监听接口的类的实例。
- 可以向事件源注册相应事件的监听器。
- 当事件发生时，事件源能够把事件对象发送给已注册的相应监听器。
- 监听器对象中的事件处理方法会使用事件对象中的信息决定对事件的反应。

Java 从 JDK1.1 开始，引入了全新的事件模型，两个事件模型之间存在着许多的区别，其中两个主要的区别如下。

(1) 在 JDK 1.0 中，控件产生的事件总是被包含该控件的对象接收到。在 JDK 1.1 以及以后的版本中，控件产生的事件则被发送到控件的事件监听器。

(2) 在 JDK 1.0 中，所有的事件都是在 `action` 和 `handleEvent` 方法中捕获的。在 JDK 1.1 以及以后的版本中，则有了许多能够响应事件的独立方法（如 `actionPerformed` 和 `itemStateChanged`）。

在本章中，重点关注用户界面事件的处理，然而基础事件处理体系结构并不仅限于用户界面事件，表 3-1 列出了经常用到的事件对象及其作用。

表 3-1 常用事件对象及其作用

事件名称	作用
<code>AWTEvent</code>	所有 AWT 事件的根事件类
<code>ConnectionEvent</code>	封装与连接有关信息的事件
<code>ChangeEvent</code>	封装状态改变信息的事件
<code>CaretEvent</code>	封装文本区中的光标改变信息的事件
<code>ListSelectionEvent</code>	表现当前选择中更改的特征事件
<code>ListDataEvent</code>	定义一个封装列表更改的事件
<code>HyperlinkEvent</code>	封装与超文本链接有关的事件
<code>TableColumnModelEvent</code>	某一个表的列模型已发生更改
<code>TableModelEvent</code>	通知侦听器某一表模型已发生更改
<code>TreeExpansionEvent</code>	用于标识树中的单个路径的事件
<code>TreeModelEvent</code>	封装描述树模型更改的信息，并用于通知侦听更改的树模型监听器
<code>TreeSelectionEvent</code>	描述当前选择的更改事件，该更改以任意数量的路径为基础

3.2 AWT 与 Swing 事件

在了解了事件处理的工作机制以后，现在深入讨论 Java 中的事件处理。Java 中的事件处理是面向对象的，所有事件都是从 `java.util` 包中的 `EventObject` 类扩展而来的。`EventObject` 有一个子类 `AWTEvent`，它是所有 AWT 事件的父类，图 3-2 为 AWT 事件的继承关系图。

对于 Java 开发人员来说，有些 AWT 事件在实际中应用不多。例如 AWT 把 `PaintEvent` 对象插进事件队列中，但是这些对象并不传递到监听器。开发人员应该重载 `paintComponent` 方法来控制重绘，表 3-2 列举了实际会被传递到监听器的 AWT 事件类型。

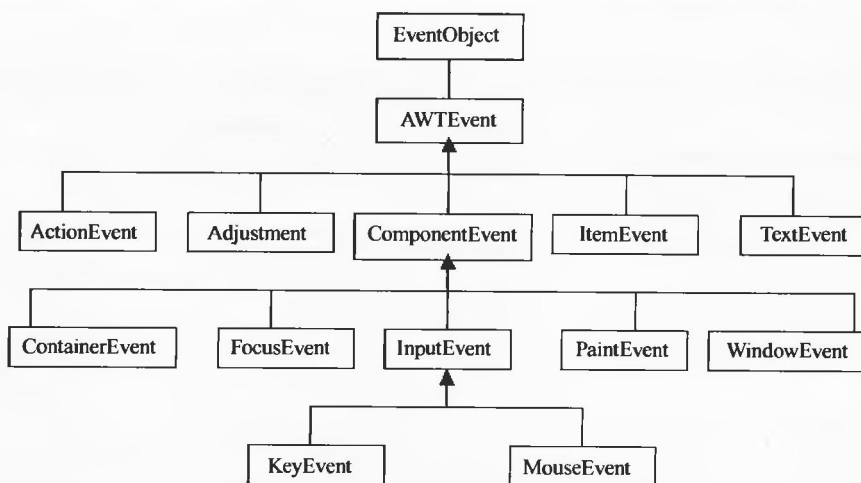


图 3-2 AWT 事件的继承关系图

表 3-2

会被传递到监听器的 AWT 事件类型

AdjustmentEvent	ComponentEvent	MouseEvent	TextEvent
ContainerEvent	FocusEvent	WindowEvent	ActionEvent
ItemEvent	KeyEvent		

对于 AWT 中的每个事件，都有相应的监听接口，以及相应的事件处理方法。表 3-3 列出了常用的 AWT 事件、监听器接口和事件处理方法。

表 3-3

常用的事件类型和接口

事件类型	接口名称	接口中声明的方法	作用
ActionEvent	ActionListener	actionPerformed(ActionEvent e)	监听组件的某个动作
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent e)	监听调整事件
ComponentEvent	ComponentListener	componentHidden(ComponentEvent e)	监听组件隐藏事件
		componentMoved(ComponentEvent e)	监听组件移动事件
		componentResized(ComponentEvent e)	监听组件调整尺寸事件
		componentShown(ComponentEvent e)	监听组件显示事件
ContainerEvent	ContainerListener	componentAdded(ContainerEvent e)	监听容器添加组件事件
		componentRemoved(ContainerEvent e)	监听容器移除组件事件
FocusEvent	FocusListener	focusGained(FocusEvent e)	监听键盘获得焦点事件
		focusLost(FocusEvent e)	监听键盘失去焦点事件
HierarchyEvent	HierarchyBoundsListener	ancestorMoved(HierarchyEvent e)	监听父窗口的移动事件
		ancestorResized(HierarchyEvent e)	监听父窗口的调整尺寸事件

续表

事件类型	接口名称	接口中声明的方法	作用
InputMethodEvent	InputMethodListener	caretPositionChanged(InputMethodEvent e)	监听输入方式改变事件
		inputMethodTextChanged(InputMethodEvent e)	
ItemEvent	ItemListener	itemStateChanged(ItemEvent e)	监听项目是否选中状态的改变
KeyEvent	KeyListener	keyPressed(KeyEvent e)	监听键盘按下事件
		keyReleased(KeyEvent e)	监听键盘释放事件
		keyTyped(KeyEvent e)	监听键盘单击事件
MouseEvent	MouseListener	mouseClicked(MouseEvent e)	监听鼠标的单击事件
		mouseEntered(MouseEvent e)	监听鼠标的移入事件
		mouseExited(MouseEvent e)	监听鼠标的移出事件
		mousePressed(MouseEvent e)	监听鼠标的按下事件
		mouseReleased(MouseEvent e)	监听鼠标的释放事件
	MouseMotionListener	mouseDragged(MouseEvent e)	监听鼠标在组件内的拖动事件
		mouseMoved(MouseEvent e)	监听鼠标在组件内的拖动事件
	MouseWheelListener	mouseWheelMoved(MouseEvent e)	监听鼠标滚轮的活动
TextEvent	TextListener	textValueChanged(TextEvent e)	监听文本类型的改变
WindowEvent	WindowFocusListener	windowGainedFocus(WindowEvent e)	监听窗口的各种一般事件 监听窗口事件
		windowLostFocus(WindowEvent e)	
	WindowListener	windowActivated(WindowEvent e)	
		windowClosed(WindowEvent e)	
		windowClosing(WindowEvent e)	
		windowDeactivated(WindowEvent e)	
		windowDeiconified(WindowEvent e)	
		windowIconified(WindowEvent e)	
		windowOpened(WindowEvent e)	
	WindowStateListener	windowStateChanged(WindowEvent e)	监听窗口状态改变事件

在表 3-3 所列出的事件中, 最常用的是动作事件 (ActionEvent)、键盘事件 (KeyEvent)、鼠标事件 (MouseEvent) 和窗口事件 (WindowEvent)。在接下来的几节中会对这些事件分别进行讨论。

有些 Swing 组件生成其他的事件类型对象, 其直接扩展 EventObject, 而不是扩展 AWTEvent。javax.swing.event 中包含了很多 Swing 组件专用的监听器接口, 但是通常对于基本的事件处理来说仍然使用基本的 AWT 监听器接口。对于 Swing 组件专用的事件以及监听器接口的使用会在以后的章节中介绍。

如果需要, 还可以通过扩展 EventObject 或者其他事件类添加定制事件。

3.3 动作事件

当用户完成对 JButton、JCheckBox、JComboBox、JTextField 或 JRadioButton 这些组件的一个操作时，将会产生一个动作事件。程序员需要实现 ActionListener 接口来处理该事件。

ActionListener 接口中只声明了 actionPerformed(ActionEvent e) 一个方法。此方法用来对事件进行处理，应该将处理事件的代码放入该方法中。

3.4 键盘事件

键盘事件，需要通过 KeyListener 接口来实现。KeyListener 接口中声明了 3 个方法，即 keyPressed(KeyEvent e)、keyReleased(KeyEvent e) 和 keyTyped(KeyEvent e)。其中 keyPressed(KeyEvent e) 和 keyReleased(KeyEvent e) 是按键被按下和松开时所触发的事件处理方法，依赖于系统和键盘的布局。keyTyped(KeyEvent e) 是与平台和布局无关的，表示按键被单击。

3.5 鼠标事件

Java 中提供了 MouseListener、MouseMotionListener 和 MouseWheelListener 3 个接口来处理各类鼠标事件。MouseListener 用于监听鼠标的单击、移入组件区域、移出组件区域的操作。MouseMotionListener 用于监听鼠标在组件上的移动或拖动操作。MouseWheelListener 用于监听鼠标滚轮。每一个接口中又声明了许多方法，表 3-3 中已经列出了这些方法，这里不再赘述。

3.6 窗口事件

Java 中提供了 WindowFocusListener、WindowListener 和 WindowStateListener 3 个接口来处理各类窗口事件。其中，WindowFocusListener 用于监听窗口焦点的获取与丢失事件；WindowListener 用于监听各种一般的窗口事件，如窗口的打开、关闭和激活等；WindowStateListener 用于监听窗口状态的改变事件。每一个接口中又声明了多种方法，表 3-3 中已经列出了这些方法，这里不再赘述。

需要特别指出的是，WindowListener 中声明的 windowClosing(WindowEvent e) 和 windowClosed(WindowEvent e) 方法看起来类似，但实质上两者是不同的。前者在窗口正在被关闭时调用，而后者在窗口关闭后被调用。实际应用时可以在 windowClosing(WindowEvent e) 方法中添加释放相应资源的代码。

3.7 事件适配器

为了进行事件处理，必须要实现某些特定的监听接口。但是某些接口，如 WindowListener 中声明了很多方法，要实现这些接口，必须实现所有声明的方法。对于开发人员来说，这无

是一项非常繁重的工作。

在许多情况下，程序中实际用到的只是监听接口中声明的个别方法，因此为了编程方便，Java 中为那些声明了多个方法的接口又提供了一个适配器类。在适配器类中实现了相应接口中的全部方法，但是该方法的具体实现代码为空。

有了适配器技术，开发人员如果不需要用到监听接口中的所有方法则可以通过继承相应的适配器类并覆盖需要的事件处理方法即可，而不必实现接口中的所有方法，这为开发提供了很大的方便。

比如共有 WindowFocusListener、WindowListener 和 WindowStateListener 3 个接口来监听窗口事件，仅其中的 WindowListener 接口就声明了 7 种方法，该接口的定义如下：

```
public interface WindowListener Extends EventListener
{
    public void windowOpened(WindowEvent e);
    public void windowClosing(WindowEvent e);
    public void windowClosed(WindowEvent e);
    public void windowIconified(WindowEvent e);
    public void windowDeiconified(WindowEvent e);
    public void windowActivated(WindowEvent e);
    public void windowDeactivated(WindowEvent e);
}
```

Java 为上面列出的 3 个接口提供了一个适配器 WindowAdapter 类，该类定义为：

```
public abstract class WindowAdapter implements
    WindowListener, WindowStateListener, WindowFocusListener
{
    public void windowOpened(WindowEvent e) {}
    public void windowClosing(WindowEvent e) {}
    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
    public void windowStateChanged(WindowEvent e) {}
    public void windowGainedFocus(WindowEvent e) {}
    public void windowLostFocus(WindowEvent e) {}
}
```

这样，在实际编程的时候，就可以定义一个新的适配器类继承 WindowAdapter 类，只重写其中某些需要用到的方法，而不必实现那些用不到的方法，从而为程序员节省大量的时间。

Java 中提供了大量的适配器类，表 3-4 给出了一些常用的适配器类。

表 3-4 常用的适配器类

适配器类	实现的接口
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener

续表

适配器类	实现的接口
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowFocusListener, WindowListener, WindowStateListener

在本章中简要地介绍了 Java 中的事件处理模型和一些常用的事件及其监听接口,对于这些技术的具体应用将在下面的章节中具体介绍。

3.8 小结

本章主要讲述了 Swing 的事件处理基础,对几种常用的 AWT 与 Swing 事件进行了详细介绍,包括动作事件、键盘事件、窗口事件等。并给出了相应事件的事件监听器接口以及事件处理方法。在以后的章节中,将结合具体的实例,讲述如何使用事件模型进行项目的开发。

第4章

Swing 常用基本控件

控件是建立程序界面的基本元素，是可视化编程的基础。控件的使用，充分地体现了当今流行的面向对象的编程思想。本章介绍了 Swing 的一些常用基本控件，如 JLabel、JButton、JTextArea 等，并结合几个具体的例子，详细地讲述了如何在 NetBeans 中使用这些控件。

4.1 Swing 控件类

除了顶层容器外，Swing 中提供的所有控件都是从抽象类 `java.swing.JComponent` 扩展而来。Swing 控件的继承关系如图 4-1 所示。

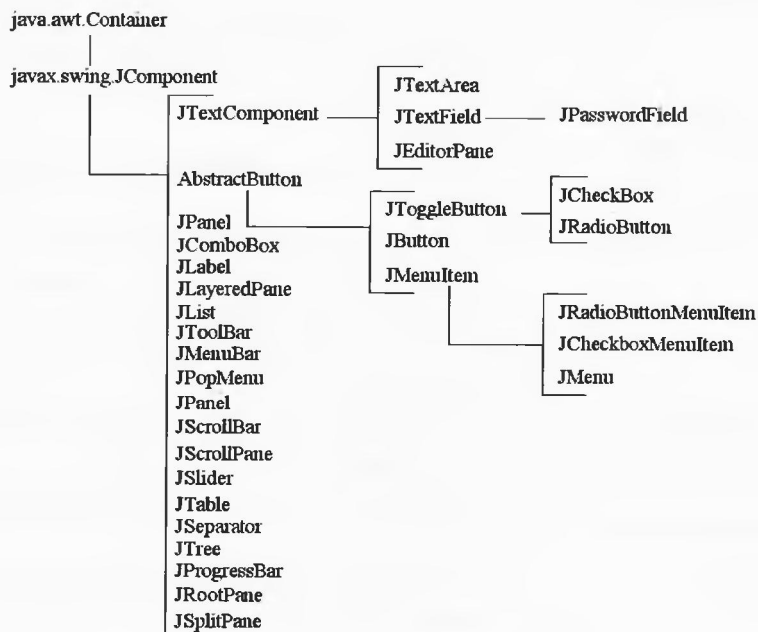


图 4-1 Swing 控件继承关系图

对于 Swing 而言, JComponent 起着导向框架的作用。JComponent 扩展了 java.awt.Container, 而 java.awt.Container 扩展了 java.awt.Component, 所以 Swing 控件拥有大量的 AWT 控件功能。

因为 JComponent 扩展了 Container, 所以 Swing 控件可以作为其他控件的容器。可以使用 add() 方法添加这些控件, 还可以在控件内部, 通过布局管理器设置被添加控件的位置大小等属性。表 4-1~表 4-6 给出了 JComponent 类中的一些常用方法。

表 4-1 定义控件外观方法及说明

方 法	说 明
getForeground()	获取当前控件的前景色
setForeground(Color c)	设置当前控件的前景色, 通常前景色用来绘制文本
getBackground()	获得当前控件的背景色
setBackground(Color c)	设置当前控件的背景色, 背景色是用来填充控件背景的颜色
setOpaque(boolean isOpaque)	设置控件是否透明
isOpaque()	获取当前控件是否是透明的
getFont()	获得当前控件的字体, 如果控件还没有设置字体, 则控件的父类字体被返回
setFont(Font f)	设置当前控件的字体
getFontMetrics(Font f)	获得特定字体的字体量度
setBorder(Border b)	设置当前控件的边框
getBorder()	获得当前控件的边框

表 4-2 绘制控件方法及说明

方 法	说 明
revalidate()	请求控件和受其影响的容器重新布局。如果不要控件可见后立即改变控件的大小, 对齐方式或者改变容器的层次关系, 则不必调用该方法
paintComponent(Graphics g)	绘制该控件
repaint(Rectangle r)	请求控件内部指定区域被重新绘制, 用 Rectangle 来指定要重绘的区域
repaint()	请求控件的所有部分重新绘制
repaint(int x,int y,int width,int height)	请求控件的指定区域被重新绘制, 4 个参数依次表示重绘区域的 x 坐标, y 坐标, 宽和高

表 4-3 设置与获取控件的绝对大小和位置

方 法	说 明
getBounds()	获取控件的位置, 返回值为 Rectangle
setBounds(int x,int y,int width,int height)	设置控件的位置, 4 个参数依次表示控件的 x 坐标, y 坐标, 宽和高
setBounds(Rectangle r)	设置控件的位置, 用 Rectangle 来指定重绘的区域
getSize()	获取控件左上角顶点的位置
setSize(int width,int height)	设置控件的大小, 参数表示控件的宽和高

续表

方 法	说 明
setSize(Dimension deme)	用指定的参数 Dimension 设置控件的大小
getLocation()	获取控件的位置, 返回值为 Dimension
setLocation(int x,int y)	设置控件顶点的位置, 参数分别代表 x 坐标和 y 坐标
setLocation(Point p)	用指定的参数 Point 设置控件顶点的位置
getLocationOnScreen()	获取控件在屏幕上的位置
getWidth()	获取控件的当前宽度
getHeight()	获取控件的当前高度
getX()	获取控件相对于其父控件左上角原点的 x 坐标
getY()	获取控件相对于其父控件左上角原点的 y 坐标
getInsets()	获取控件边框的大小

表 4-4 控件在父控件中放置的相关方法及说明

方 法	说 明
setLayout(LayoutManager lm)	设置容器的布局管理器
getLayout()	获取容器的布局管理器
applyComponentOrientation(ComponentOrientation co)	设置容器和容器中所有控件的 ComponentOrientation 属性
getAlignmentX()	获得控件在 x 轴方向的对齐
setAlignmentX(float f)	设置控件在 x 轴方向的对齐, 参数是 0~1 之间的值
getAlignmentY()	获得控件在 y 轴方向的对齐
setAlignmentY(float f)	设置控件在 y 轴方向的对齐
getPreferredSize()	获取控件的首选大小
setPreferredSize(Dimension preferredSize)	设置控件的首选大小
getMinimumSize()	获取控件的最小的大小
setMinimumSize(Dimension minimumSize)	设置控件的最小的大小
getMaximumSize()	获取控件的最大的大小
setMaximumSize(Dimension maximumSize)	设置控件的最大的大小

表 4-5 设置与获取容器的层次关系

方 法	说 明
add(Component c)	将指定的控件添加到容器的结束位置
add(Component c, int x)	将指定的控件添加到容器中有 x 指定的位置
remove(int x)	从容器中删除有 x 指定的位置处的控件
remove(Component c)	移除容器内的指定控件
removeAll()	移除容器内的所有控件
getComponetCount()	获取容器中控件的数量
getComponet(int i)	获取有 x 指定位置的控件
getComponets()	获取容器中所有的控件

表 4-6

设置与获取控件的状态

方法	说明
<code>setName(String name)</code>	设置控件的名称
<code>getName()</code>	获取控件的名称
<code>setEnabled(boolean b)</code>	设置控件的可编辑性
<code>isEnabled()</code>	获取控件的可编辑性
<code>setVisible(boolean b)</code>	设置控件的可视性
<code>isVisible()</code>	获取控件的可视性
<code>getToolTipText()</code>	获取在工具提示中显示的文本
<code>setToolTipText(String text)</code>	获取在设置要在工具提示中显示的文本

每一个 Swing 控件还有其特有的方法, 属性以及作用, 后面的几节将为读者介绍 Swing 中常用的一些控件及其使用方法。

4.2 Swing 标签

Swing 允许通过 JLabel 控件建立可以包含文本、图像, 或是两者都包含的标签。本节中主要介绍如下内容:

- JLabel 类简介;
- 在 JLabel 上使用图像;
- NetBeans 中 JLabel 的使用。

4.2.1 JLabel 类简介

JLabel 类直接扩展了 javax.swing.JComponent。下面介绍 JLabel 的一些特性。

- JLabel 是一种可以包含文本或者图像的非交互控件, 这种控件没有标签修饰 (如没有边界), 不能响应用户的输入。
- JLabel 可以由文本或者图像构成, 对于只包含文本的简单 JLabel, 类似于 java.awt.Label。
- JLabel 主要用来标识其他控件, 对于已经创建的 JLabel, 只需调整该 JLabel 的位置, 使其与被标识控件足够接近即可。

在本小节中, 主要讨论包含文本的 JLabel, 对于包含图像的情况, 将会在下面的小节中详细地介绍。创建只包含文本的 JLabel 非常简单, 如下面的代码创建了一个包含 “Hello World!” 文本的 JLabel 控件:

```
JLabel helloJLabel=new JLabel("Hello World!");
```

JLabel 类有 5 个构造器, 如表 4-7 所示。

表 4-7

JLabel 类构造器及说明

构造器	说 明
<code>JLabel()</code>	创建无图像并且其标题为空字符串的 JLabel

续表

构造器	说 明
JLabel(Icon image)	创建具有指定图像的 JLabel 实例
JLabel(String text)	创建具有指定文本的 JLabel 实例
JLabel(String text, Icon icon, int horizontalAlignment)	创建具有指定文本、图像和水平对齐方式的 JLabel
JLabel(String text, int horizontalAlignment)	创建具有指定文本和水平对齐方式的 JLabel

JLabel 能够使用多种方法来进行相应操作，常用方法和说明如表 4-8 所示。

表 4-8 JLabel 中常用的方法及其说明

方 法	说 明
getText()	获取该标签所显示的文本字符串
setText(String text)	定义此组件将要显示的单行文本，如果 text 值为 null 或空字符串，则什么也不显示，此属性的默认值为 null
getIcon()	获取该标签显示的图形图像（字形、图标）
setIcon(Icon icon)	定义此组件将要显示的图标，如果 icon 值为 null，则什么也不显示
getDisabledIcon()	获取该标签被禁用时所使用的图标，如果尚未设置禁用图标，则此方法将转发对外观的调用，以构造一个合适的禁用图标
setDisabledIcon(Icon disabledIcon)	设置如果此 JLabel 是“禁用的”（JLabel.setEnabled(false)），则要显示的图标
getVerticalAlignment()	获取标签内容沿 y 轴的对齐方式
setVerticalAlignment(int alignment)	设置标签内容沿 y 轴的对齐方式，此属性的默认值为 CENTER
getHorizontalAlignment()	获取标签内容沿 x 轴的对齐方式
setHorizontalAlignment(int alignment)	设置标签内容沿 x 轴的对齐方式
getVerticalTextPosition()	获取标签中的文本的垂直相对位置，此垂直相对位置是相对标签图像而言的
setVerticalTextPosition(int textPosition)	设置标签的文本的垂直相对位置，此垂直相对位置是相对标签图像而言的，此属性的默认值为 CENTER
getHorizontalTextPosition()	获取标签中的文本的水平相对位置
setHorizontalTextPosition(int textPosition)	设置标签的文本的水平相对位置
getLabelFor()	获取将标签添加到的组件
setLabelFor(Component c)	设置将标签添加到的组件，如果无法为 Component 添加标签，则返回值可以为 null
paramString()	获取此 JLabel 的字符串表示形式。此方法仅在进行调试的时候使用，对于各个实现，所返回字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null

提示

horizontalAlignment 和 verticalAlignment 用来指定标签内容（文本和图标）的内部对齐情况。这些属性的值是在 SwingConstants 中定义的，对于 horizontalAlignment 属性，其值必须是 LEADING、TRAILING、LEFT、RIGHT 或者 CENTER。对于 verticalAlignment 属性，其值必须是 TOP、BOTTOM 或者 CENTER。

4.2.2 在 JLabel 中使用图像

在前面的小节中已经提到, JLabel 可以由图像构成。通过使用 JLabel, 将图像添加到用户界面变得非常容易, 下面的代码创建了一个包含图像 title.jpeg 的 JLabel:

```
ImageIcon myIcon=new ImageIcon("title.jpeg");
JLabel myJLabel=new JLabel(myIcon);
```

可以看到, 创建带图像的 JLabel 并不复杂。其中关于 ImageIcon 类, 在后面的章节中会进行详细介绍。

setVerticalTextPosition(int textPosition)与 setHorizontalTextPosition(int textPosition)方法用来设置 JLabel 中文本与图像的相对位置, 其值为上一节中讨论的 SwingConstants 类中定义的值。文本与图像的相对位置不同于文本的对齐方式, 文本的对齐指文本与边框的对齐情况, 通过 setHorizontalAlignment(int alignment)和 setVerticalAlignment(int alignment)方法进行设置。

绝大多数 Swing 控件支持使用 HTML, 可以使用 HTML 标记语言设置 JLabel 中的字体和显示样式等, 如 JLabel myLabel=new JLabel("<HTML><p><h1>姓名<h1></p></HTML>")。


4.2.3 在 NetBeans 中使用 JLabel

NetBeans 中提供了功能非常强大的可视化开发工具——Matisse, 用于开发用户界面, 使得有关用户界面的设计更加快捷方便。在本小节中, 将通过一个小例子介绍如何在 NetBeans 中使用 JLabel。

(1) 首先创建一个名称为 container, 主类名为 org.netbeans.swing.container.Demo 的项目, 并向其中添加一个使用 JFrame Form 模版创建的 DemoJLabelFrame 类。

(2) 添加完成以后, 打开 GUI 设计器, 在“palette”窗口中选择“JLabel”, 之后在 GUI 设计器中的适当位置单击, 即可向 DemoJLabelFrame 添加一个 JLabel。

(3) 选中新添加的 JLabel, 单击鼠标右键, 在弹出的菜单中选择“Change Variable Name”选项, 在弹出的对话框中将该 JLabel 的名称修改为 jLabelImage。

(4) 再右键单击 JLabel, 选择右键菜单中的“Properties”选项, 将 text 属性的值设置为“演示图片”。单击 icon 属性右侧的  按钮, 弹出如图 4-2 所示的对话框。

(5) 在如图 4-2 所示的窗口中选择文件来源类型, 并通过单击“Select File”按钮选择文件。此时选中的文件会出现在“Preview”列表框中。单击“OK”按钮完成设置。此时 GUI 设计器中内容如图 4-3 所示。

(6) 在 GUI 设计器中选中 jLabelImage, 单击鼠标右键, 在弹出的菜单中选择“Events”/“Mouse”/“mouseClicked”选项, 为 jLabelImage 添加一个单击事件处理方法, 此时光标会定位到源代码编辑器如图 4-4 所示的方法内。

(7) 图 4-4 中的方法用来处理鼠标单击事件, 通过向其中添加如下代码, 以设置鼠标单击 jLabelImage 后其文本相对其图像的垂直位置为 TOP。

```
this.jLabelImage.setVerticalTextPosition(SwingConstants.TOP);
```

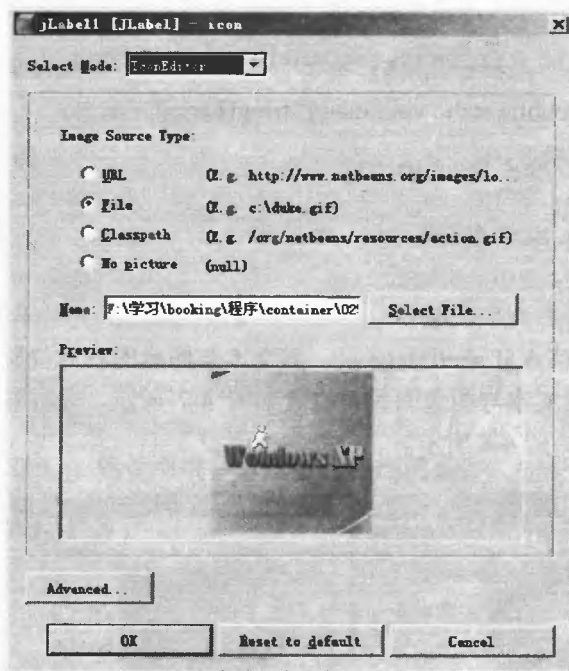


图 4-2 设置图像对话框



图 4-3 包含图像与文本的 JLabel

```
private void jLabelImageMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here.
}
```

图 4-4 鼠标单击事件处理方法



提示

将该段代码添加到方法后，源代码编辑器中会提示需要导入 `javax.swing.SwingConstants` 类，具体的导入方法在前面的章节已经讲解，这里不再赘述。

(8) 重复以上步骤，为 `JLabelImage` 分别添加 `MouseExited` 和 `MouseEntered` 事件处理方法。添加完毕后，在代码编辑器中生成如图 4-5 所示的内容。

```
private void jLabelImageMouseExited(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here.
}

private void jLabelImageMouseEntered(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here.
}
```

图 4-5 鼠标移入和移出 JLabelImage 的事件处理方法

(9) 图 4-5 中的 `jLabelImageMouseEntered(java.awt.event.MouseEvent evt)` 方法用来处理鼠标移入事件，向其中添加如下代码，以设置鼠标移入 `JLabelImage` 后其文本相对图像的垂直位置为 `BOTTOM`。

```
this.jLabelImage.setVerticalTextPosition(SwingConstants.BOTTOM);
```

(10) 图 4-5 中的 `jLabelImageMouseExited(java.awt.event.MouseEvent evt)` 方法用来处理鼠标移出事件，向其中添加如下代码，以设置鼠标移出 `JLabelImage` 后其文本相对其图像的

垂直位置为 Center。

```
this.jLabelImage.setVerticalTextPosition(SwingConstants.Center);
```

(11) 将下列代码添加到 Demo 类中的 public static void main(String[] args)方法内:

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoJLabelFrame().setVisible(true);  
    }  
});
```

(12) 编译并运行主项目, 会看到鼠标移入 jLabelImage 后, 其文本与图像的相对位置如图 4-6 所示。在 jLabelImage 上单击后, 其文本与图像的相对位置如图 4-7 所示。光标移出 jLabelImage 后, 其图像与文本的位置关系如图 4-8 所示。



图 4-6 鼠标移入后窗口图

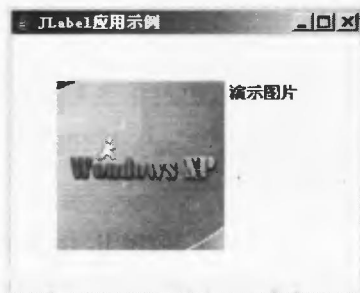


图 4-7 单击鼠标后窗口图



图 4-8 鼠标移出后窗口图



提示

利用鼠标在标签上移入和移出, 可以制作功能非常强大的弹出式菜单。具体的制作方法, 会在以后的章节中介绍。

4.3 Swing 按钮

JButton 是 Swing 的常用控件, 是图形用户界面编程的重要组成部分。本节将对 JButton 进行介绍, 主要包括以下内容:

- JButton 类简介;
- NetBeans 中 JButton 的使用。

4.3.1 JButton 类简介

JButton 扩展自 javax.swing.AbstractButton, 是 Swing 中最简单的按钮类型, 其中可以包含文本或图标, 能够响应单击事件。创建 JButton 对象非常简单, 下面是用于创建一个包含“确定”文本按钮的代码:

```
JButton myButton=new JButton("确定");
```

下面的代码创建了一个包含“buttonicon.gif”图标的按钮。

```
JButton myButton=new JButton("buttonicon.gif");
```

JButton 类有 5 个构造器，如表 4-9 所示。

表 4-9 JButton 类构造器及其说明

构造器	说 明
JButton()	创建不带有设置文本或图标按钮
JButton(Action a)	创建一个按钮，其动作属性从所提供的 Action 中获取
JButton(Icon icon)	创建一个带图标的按钮
JButton(String text)	创建一个带文本的按钮
JButton(String text, Icon icon)	创建一个带初始文本和图标的按钮

JButton(Action a)构造器是在 JDK 1.3 中引入的，其创建了一个来自指定 Action 属性值的按钮，注册该 Action，以便接收按钮触发的 ActionEvent。

JButton 能使用很多方法进行相应的操作，如表 4-10 所示。

表 4-10 JButton 的常用方法及其说明

方 法	使用 说明
addActionListener()	为按钮添加 ActionEvent 事件监听器
removeActionListener(ActionListener l)	从按钮中移除一个 ActionListener。如果监听器是按钮当前设置的 Action，则将 Action 设置为 null
getAction()	获取当前为 ActionEvent 源设置的 Action，如果没有设置任何 Action，则返回 null
getModel()	获取此按钮表示的模型
setModel(ButtonModel newModel)	设置此按钮表示的模型
getMargin()	获取值为指定按钮边框和标签之间的空白的 Insets 对象
setMargin(Insets m)	设置按钮边框和标签之间的空白。将该空白设置为 null 会造成按钮使用默认空白
getSelectedIcon()	获取按钮被选中时的图标
setSelectedIcon(Icon selectedIcon)	设置按钮被选中时的图标

上一节中介绍的 getHorizontalAlignment()、setHorizontalAlignment(int alignment)、getVerticalTextPosition()和 setVerticalTextPosition(int textPosition)4 个方法，在 JButton 中也同样适用。使用方法与在 JLabel 中一致，这里不再赘述。

JButton 还能够使用其父类中提供的很多方法，如 setIcon(ImageIcon icon)、setText(String text)等方法，这里并没有全部给出，如果需要，用户可以查看 API。

JButton 类中没有定义新的事件，从父类继承的事件中，最重要的事件是单击 JButton 按钮时所触发的 ActionEvent 事件。该事件在按钮被释放后，并且释放时鼠标必须仍然在按钮上才能触发该事件。

4.3.2 NetBeans 中 JButton 的使用

在 NetBeans 中使用 JButton 非常方便，本小节将通过一个例子介绍如何在 NetBeans 中使用 JButton。

(1) 打开上一节中创建的项目 container，然后向其中添加一个使用“JFrame Form”模版

创建的 DemoJButtonFrame 类。

(2) 添加完成以后, 打开 GUI 设计器, 在 “palette” 窗口中选择 “JButton” 选项, 之后在 GUI 设计器中的适当位置单击鼠标左键, 即可向 DemoJButtonFrame 类添加一个 JButton。

(3) 右键单击该 JButton, 选择快捷菜单中的 “Change Variable Name” 选项, 在弹出的对话框中将该 JButton 的名称修改为 jButtonEnter。

(4) 按照上述步骤再向 DemoJButtonFrame 类添加一个名称为 jButtonCancel 的 JButton。接着向 DemoJButtonFrame 添加一个名称为 jLabelMessage 的 JLabel, 用来输出一些提示信息。

(5) 添加控件完成以后, 按照如表 4-11 所示修改控件属性。

表 4-11 控件与属性值对照表

控 件 名 称	text	horizontalAlignment
jLabelMessage	用户还没有按下按钮	CENTER
jButtonEnter	确定	默认值
jButtonCancel	取消	默认值

(6) 此时 GUI 设计器中内容如图 4-9 所示。

(7) 在如图 4-9 所示的 GUI 设计器中的 “确定” 按钮上单击鼠标右键, 依次选择菜单中的 “Events” / “Action” / “actionPerformed” 选项, 为 “确定” 按钮添加 ActionEvent 事件的处理方法。

(8) 按照同样的方法, 为 “取消” 按钮添加 ActionEvent 事件处理方法。此时, 源代码编辑器中会生成如图 4-10 所示的两个方法。



图 4-9 按钮与标签

```
private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here  
}  
  
private void jButtonEnterActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here  
}
```

图 4-10 按钮的 ActionEvent 事件处理方法

- jButtonCancelActionPerformed(java.awt.event.ActionEvent evt)方法用来处理 “取消” 按钮的 ActionEvent 事件。
- jButtonEnterActionPerformed(java.awt.event.ActionEvent evt)方法用来处理 “确定” 按钮的 ActionEvent 事件。

(9) 将下列代码添加到 jButtonCancelActionPerformed(java.awt.event.ActionEvent evt)方法中, 当单击 “取消” 按钮时, jLabelMessage 标签显示 “你单击了取消按钮”。

```
this.jLabelMessage.setText("你单击了取消按钮");
```

(10) 将下列代码添加到 jButtonEnterActionPerformed(java.awt.event.ActionEvent evt)方法中, 当单击 “确定” 按钮时, jLabelMessage 显示 “你单击了确定按钮”。

```
this.jLabelMessage.setText("你单击了确定按钮");
```

(11) 将如下代码添加到 Demo 类的 public static void main(String[] args)方法中, 并删除

原来方法中的代码。

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoJButtonFrame().setVisible(true);  
    }  
});
```

编译并运行主项目，窗口如图 4-11 所示。单击“确定”按钮后，窗口如图 4-12 所示。单击“取消”按钮后，窗口如图 4-13 所示。

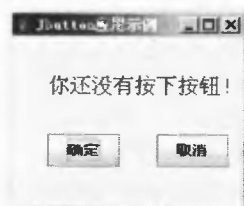


图 4-11 初始效果图

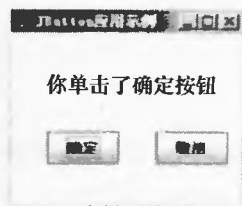


图 4-12 单击确定按钮效果图

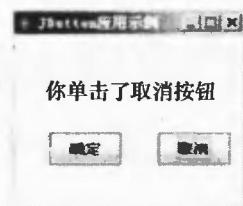


图 4-13 单击取消按钮效果图

4.4 Swing 文本框

JTextField 也是图形用户界面编程中经常用到的控件。本节中将对 JTextField 进行介绍，具体内容如下：

- JTextField 类简介；
- NetBeans 中 JTextField 的使用。

4.4.1 JTextField 类简介

JTextField 扩展自 javax.swing.text.JTextComponent，用于输入单行的文本，如果文本的长度超过字段的实际长度，它会自动滚动文本。下面的代码创建了一个初始内容为“你好，这是 JTextField”、宽度为 60 的 JTextField。

```
JTextField jTextFieldOne=new JTextField("你好，这是 JTextField");
```

下面的代码创建了一个初始内容为空，宽度为 60 个字符的 JTextField。

```
JTextField jTextFieldTwo=new JTextField(60);
```

JTextField 有 5 个构造器，如表 4-12 所示。

表 4-12

JTextField 构造器及简单说明

构造器	说明
JTextField()	构造一个新的 JTextField。创建一个默认模型，初始字符串为 null，列数设置为 0
JTextField(Document doc, String text, int columns)	构造一个新的 JTextField，它使用给定文本存储模型和给定的列数

续表

构造器	说 明
<code>JTextField(int columns)</code>	构造一个具有指定列数的新的空 <code>JTextField</code> , 创建默认模型, 初始字符串设置为 <code>null</code>
<code>JTextField(String text)</code>	构造一个用指定文本初始化的新 <code>JTextField</code> , 创建列数为 0 的默认模型
<code>JTextField(String text, int columns)</code>	构造一个用指定文本和列初始化的新 <code>JTextField</code> , 创建默认模型

对 `JTextField` 进行操作的方法有多种, 具体如表 4-13 所示。

表 4-13 `JTextField` 常用的方法及说明

方 法	说 明
<code>setDocument(Document doc)</code>	将编辑器与一个文本文档关联。当前注册的工厂用来生成文档的一个视图, 此视图经过重新验证之后由编辑器来显示
<code>createDefaultModel()</code>	如果在创建 <code>JTextField</code> 对象时没有明确地给出要使用的文档模型, 则会调用此方法返回默认的文档模型对象, 其为 <code>PlainDocument</code> 类的实例
<code>getColumns()</code>	获取此 <code>JTextField</code> 中的列数
<code>setColumns(int columns)</code>	设置此 <code>JTextField</code> 中的列数, 使用该方法后, 应该调用容器的 <code>validate()</code> 方法来重新绘制组件
<code>setFont(Font f)</code>	设置当前字体。这将移除缓存的行高和列宽, 以便新的字体能够反映出来, 设置字体后将调用 <code>revalidate</code>
<code>addActionListener(ActionListener l)</code>	添加指定的操作侦听器, 以/从此文本字段接收操作事件
<code>removeActionListener(ActionListener l)</code>	移除指定的操作侦听器, 以便不再从此文本字段接收操作事件
<code>getText()</code>	获取此 <code>JTextField</code> 中包含的文本
<code>setText(String t)</code>	设置 <code>JTextField</code> 中显示的文本

另外, 从父类中继承的 `getHorizontalAlignment()` 与 `setHorizontalAlignment(int alignment)` 方法用来设置文本的水平对齐方式, 有效值包括 `JTextField.LEFT`、`JTextField.CENTER`、`JTextField.RIGHT`、`JTextField.LEADING` 和 `JTextField.TRAILING`。

4.4.2 NetBeans 中 `JTextField` 的使用

在 NetBeans 中创建 `JTextField` 非常方便, 本小节将通过一个例子展示如何在 NetBeans 中使用 `JTextField`。具体步骤如下。

(1) 在 NetBeans 中打开前面创建的项目 `container`, 然后向其中添加一个使用“`JFrame Form`”模板创建的 `DemoJTextFieldFrame` 类。

(2) 添加完成以后, 打开 GUI 设计器, 在“`Palette`”窗口中选择 `JTextField` 选项, 之后在 GUI 设计器中的适当位置单击鼠标左键, 即可向 `DemoJTextFieldFrame` 类添加一个 `JTextField`。

(3) 右键单击 `JTextField`, 在弹出的菜单中选择“`Change Variable Name`”选项, 接着在弹出的对话框中将 `JTextField` 的名称修改为 `jTextFieldTest`。并在属性对话框中修改 `jTextFieldTest` 的属性, 将 `text` 的值设置为空。

(4) 向 DemoJTextFieldFrame 添加一个 JLabel, 用其显示在 jTextFieldTest 中输入的内容。将该 JLabel 的名字修改为 jLabelMessage, 并将其 text 属性值修改为“你还没有输入任何内容”。

(5) 在 GUI 设计器中的“确定”按钮上右键单击该控件, 在弹出的菜单中依次选择“Events”/“Action”/“actionPerformed”选项, 为 jTextFieldTest 添加 ActionEvent 事件的处理方法。此时, 源代码编辑器中会生成如图 4-14 所示的方法。

```
private void jTextFieldTestActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

图 4-14 JTextField 处理 ActionEvent 事件的方法

提示 jTextFieldTestActionPerformed(java.awt.event.ActionEvent evt)方法用来处理 jTextFieldTest 文本框的 ActionEvent 事件。

(6) 将下列代码添加到 jTextFieldTestActionPerformed(java.awt.event.ActionEvent evt)方法的方法中。

```
String getMessage=this.jTextFieldTest.getText().trim();
if(getMessage.equals("")){
    this.jLabelMessage.setText("你还没有输入任何内容");
}
else{
    this.jLabelMessage.setText("你输入的内容是:"+getMessage);
}
```

上面代码的作用如下。

- 如果用户输入非空内容后回车, 则标签上显示“你输入的内容是:”与用户输入的内容。
- 如果用户没有输入内容或输入空内容(如空格、退格等)后按下回车, 则标签上显示“你还没有输入任何内容”。

(7) 将如下代码添加到 Demo 类的 public static void main(String[] args)方法中, 并删除原来方法中的代码。

```
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoJTextFieldFrame().setVisible(true);
    }
});
```

(8) 编译并运行主项目, 在没有输入任何内容时按下回车, 窗口如图 4-15 所示。在 jTextFieldTest 中输入“你好”后回车, 此时窗口如图 4-16 所示。

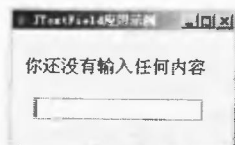


图 4-15 没有输入内容时窗口

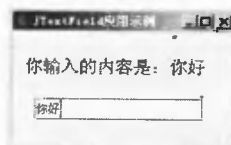


图 4-16 输入内容后窗口

JButton 与 JTextField 都可以触发 ActionEvent 事件, 所不同的是, JButton 的 Action-Event 事件在单击按钮时被触发, 而 JTextField 的 ActionEvent 事件在按下回车键时被触发。

4.5 格式化文本框

JFormattedTextField 是一种非常有用的控件, 其可以根据所在地域的不同自动改变显示格式, 可以非常方便地设置日期的显示格式, 限制值的范围, 使值通过键盘来进行增减等特性。本节将对 JFormattedTextField 展开讨论, 主要包括如下内容:

- JFormattedTextField 类简介;
- NetBeans 中 JFormattedTextField 的使用。

4.5.1 JFormattedTextField 类简介

JDK 1.4 中引入了 JFormattedTextField 类, 其扩展了 JTextField, 为开发人员提供了指定输入到一个文本框的合法字符集的方法, 下面列出了 JFormattedTextField 的一些特性。

- 对格式化任意值的支持。
- 一旦用户编辑了文本就立即检索特定对象。
- 增加了一个 Value 属性, 在用户提交编辑的内容之前, Value 的值不会改变。如果用户取消了编辑, 则 JFormattedTextField 的值会恢复为 Value 的值。

以下代码说明了如何配置 JFormattedTextField 的 value 属性来编辑数值:

```
JFormattedTextField myFTF= new JFormattedTextField();  
myFTF.setValue(965325112);
```

JFormattedTextField 允许配置在失去焦点时应该使用的动作, 如表 4-14 所示。

表 4-14 失去焦点时使用的动作及说明

值	说 明
JFormattedTextField.REVERT	恢复显示以匹配 getValue, 这可能丢失当前的编辑内容
JFormattedTextField.COMMIT	提交当前值。如果 AbstractFormatter 不认为所编辑的值是合法值, 则抛出 ParseException, 然后不更改该值并保留已编辑的值
JFormattedTextField.COMMIT_OR_REVERT	与 COMMIT 类似, 但是如果该值不是合法的, 则其行为类似于 REVERT
JFormattedTextField.PERSIST	不执行任何操作, 不获取新的 AbstractFormatter 也不更新该值

JFormattedTextField 有 6 个构造器, 其中主要 4 个构造器具体说明如表 4-15 所示。

表 4-15 JFormattedTextField 构造器及说明

构 造 器	说 明
JFormattedTextField()	创建一个没有 AbstractFormatterFactory 的 JformattedTextField, 使用 setMask 或 setFormatterFactory 配置 JFormattedTextField 以编辑特定的值类型

续表

构造器	说 明
JFormattedTextField(Object value)	创建一个具有指定值的 JFormattedTextField, 这将根据 Value 的类型创建一个 AbstractFormatterFactory
JFormattedTextField(Format format)	创建一个 JFormattedTextField, format 包装在一个适当的 AbstractFormatter 中, 然后该 AbstractFormatter 包装在一个 AbstractFormatterFactory 中
JFormattedTextField(JFormattedTextField, AbstractFormatter formatter)	创建一个具有指定 AbstractFormatter 的 JFormattedTextField, 该 AbstractFormatter 被放置在 AbstractFormatterFactory 中

JFormattedTextField 继承了 JTextField, 可以使用很多方法对 JFormattedTextField 进行操作, 具体如表 4-16 所示。

表 4-16 JFormattedTextField 类的常用方法及说明

方 法	说 明
setFocusLostBehavior(int behavior)	设置丢失焦点时的行为, 参数为表 4-8 中列出的值
getFocusLostBehavior()	获取丢失焦点时的行为
setFormatter(JFormattedTextField.AbstractFormatter formatter)	设置当前的 AbstractFormatter, 通常不应调用此方法, 而是设置 AbstractFormatterFactory 或设置值
getFormatter()	获取用于格式化和分析当前值的 AbstractFormatter
setValue(Object value)	设置值, 该值由从当前 AbstractFormatterFactory 获得的 AbstractFormatter 进行格式化, 如果尚未指定任何 AbstractFormatterFactory, 则此方法会试图根据 Value 的类型创建它
getValue()	获取最后一个有效值, 根据 AbstractFormatter 的编辑策略, 此方法可能不会返回当前值, 可通过调用 commitEdit 后调用 getValue 来获得当前编辑的值
invalidEdit()	用户输入无效值时调用, 这就使得组件可以提供反馈, 默认的实现是发出蜂鸣声

可以使用 getText() 或者 setText(String s) 方法对 JFormattedTextField 的 text 属性进行设置, 但 text 属性与 Value 属性是两个不同的属性, 主要有如下不同点。

- text 属性属性由 JTextField 定义, 通常反映 JFormattedTextField 所显示的值, 而 Value 属性可能无法反映 JFormattedTextField 中显示的内容。
- 当用户输入数据时, text 属性将随之发生变化, 而 Value 属性只有被提交之后才能更新, Value 属性值通常要滞后 text 属性值。

JFormattedTextField 类不触发任何新的事件类型, 其从 JTextField 继承事件, 并在 Value 属性或者 Formatter 属性的值改变时触发属性变化事件。

从表 4-15 中可以看出, 使用指定的格式器创建 JFormattedTextField, 下面给出了两种常用的格式器:

- MaskFormatter;
- NumberFormatter。

MaskFormatter 能够指定 JFormattedTextField 的有效字符格式, 下面的代码显示了如何用 MaskFormatter 创建一个允许用户输入 8 位电话号码的 MaskFormatter:

```
phoneNumber=new JFormattedTextField(getFormatter("#####"));
private MaskFormatter getFormatter(String s){
    MaskFormatter mFormatter=null;
    try{
        mFormatter=new MaskFormatter(s);//其中 s 为掩码字符串
    }
    catch(java.text.ParseException e){
        System.out.println(e.getMessage());
    }
    return mFormatter;
}
```

MaskFormatter 使用掩码字符串来对输入字符串进行格式化检查。掩码字符串是由字符和匹配符构成的。字符只匹配自身，单引号加在匹配符之前将其转换成字符。掩码匹配符具体如表 4-17 所示。

表 4-17 掩码匹配符

字 符	匹 配
*	任意字符
A	任何字符或数字
#	任何有效的数字
'	Escape 字符，用来替换任何特殊格式字符
U	任何字符或数字，所有的字符都变成大写字符
L	任何字符或数字，所有字符都变成小写
?	任何字符
H	任何十六进制的字符（0~9、a~f、A~F）

NumberFormatter 使用 `java.text.NumberFormat` 作为其对象类型。如果在使用 `JFormattedTextField` 构造函数时，使用了 `Number` 的子类或 `java.text.NumberFormat`，那么构造函数将创建一个 `NumberFormat` 的实例。

下面代码演示了如何使用 `NumberFormatter` 创建一个初始值为 99.99 的 `JFormattedTextField`，具体设置如下：

```
myFormattedTextField=new JFormattedTextField(java.text.NumberFormat.
getInstance());
myFormattedTextField.setValue(new Float(99.99));
```

4.5.2 NetBeans 中 JFormattedTextField 的使用


在 NetBeans 中，可以更加方便地创建 `JFormattedTextField`。本小节将通过一个完整的例子介绍如何在 NetBeans 中创建 `JFormattedTextField`。

打开前面创建的项目 `container`，按照如下步骤完成该例。

(1) 向 `container` 项目中添加一个通过 `JFrame Form` 模版创建的 `DemoJFormattedTextField-Frame` 类。

(2) 添加完成以后, 打开 GUI 设计器, 在 “palette” 窗口中选择 “JFormattedTextField” 选项, 之后在 GUI 设计器中的适当位置单击, 即可向 DemoJFormattedTextFieldFrame 中添加一个 JFormattedTextField。

(3) 在 GUI 设计器中选中新添加的 JFormattedTextField 控件, 右键单击该控件, 选择右键菜单中的 “Change Variable Name” 选项。在弹出的对话框中, 将 JFormattedTextField 的名称修改为 jFormattedTextFieldNumber。并在属性对话框中修改 jFormattedTextFieldNumber 的 text 属性, 将其值设置为空。

(4) 在 GUI 设计器中选中步骤 (2) 中添加的 JFormattedTextField, 单击鼠标右键, 在弹出的菜单中选择 “Properties” 选项, 打开 jFormattedTextFieldNumber 的属性对话框, 单击属性对话框中的 “Code” 按钮, 在此窗口中单击 Custom Creation Code 右侧的  按钮, 在如图 4-17 所示的窗口中修改 Custom Creation Code 属性。

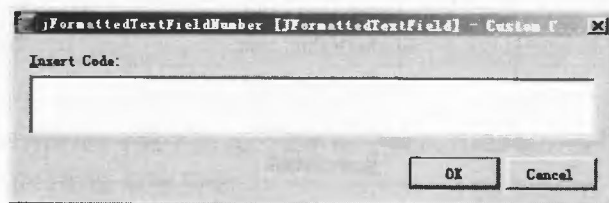


图 4-17 设置 Custom Creation Code 属性

(5) 将下列代码添加到图 4-17 所示的文本框中, 作为 jFormattedTextFieldNumber 的初始化代码。

```
new JFormattedTextField( java.text.NumberFormat.getInstance());
```

(6) 单击 “OK” 按钮, 此时即可成功地向 DemoJFormattedTextFieldFrame 中添加了一个使用 java.text.NumberFormat 实例作为格式器的 JFormattedTextField。

(7) 向 DemoJFormattedTextFieldFrame 添加一个名称为 getFormatter、参数类型为 String、参数值为 s、返回值为 MaskFormatter、访问限制修饰符为 private 的方法。该方法通过给定的字符创建一个 MaskFormatter 并返回。将下列代码添加到该方法的方法中:

```
MaskFormatter mFormatter=null;
try{
    mFormatter=new MaskFormatter(s);
}catch(java.text.ParseException e){
    System.out.println(e.getMessage());
}
return mFormatter;
```



提示

添加该段代码的时候, 会提示没有导入 javax.swing.text.MaskFormatter 类。根据提示导入该类即可。

(8) 向 DemoJFormattedTextFieldFrame 中再添加一个名称为 jFormattedTextFieldMask 的 JFormattedTextField, 并将其 text 属性设置为空, 并按照步骤 (4)、(5), 设置如下代码为 jFormattedTextFieldMask 的初始化代码段:


```
new JFormattedTextField(this.getFormatter("HHHHHHHH"));
```

提示 "HHHHHHHH"是掩码字符串，表示长度为8的16进制数字序列。

(9) 此时，就成功地向 DemoJFormattedTextFieldFrame 添加了一个使用 javax.swing.text.MaskFormatter 作为格式器的 JFormattedTextField。

(10) 依次向 DemoJFormattedTextFieldFrame 添加 6 个 JLabel，并根据如表 4-18 所示的值修改其属性。

表 4-18 属性与值对照表		
控件类型	控件名称	text 属性值
JLabel	jLabelTextName	当前 text 的值是：
JLabel	jLabelTextValue	965325418
JLabel	jLabelValueName	当前 value 的值：
JLabel	jLabelValueValue	965325418
JLabel	jLabelNumber	数字格式：
JLabel	jLabelMask	掩码格式：

(11) 向 DemoJFormattedTextFieldFrame 添加一个名称为 initFormatter、返回值为 void、访问限制修饰符为 private 的方法。将下列代码添加到该方法的方法中：

```
this.jFormattedTextFieldNumber.setValue(new Double(965325418));
this.jFormattedTextFieldNumber.setColumns(10);
this.jFormattedTextFieldMask.setValue("abc123ABC");
```

提示 该方法用来设置 jFormattedTextFieldNumber 与 jFormattedTextFieldMask 的初始值。

(12) 此时，界面如图 4-18 所示。

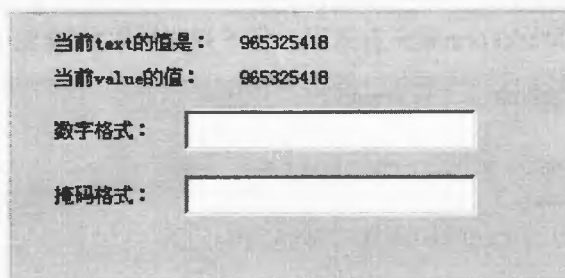


图 4-18 程序界面

(13) 用鼠标右键单击 jFormattedTextFieldNumber，依次选择菜单中的“Events”/“PropertyChange”选项，为 jFormattedTextFieldNumber 添加 PropertyChange 事件的处理方法。

(14) 将如下代码添加到 PropertyChange 事件的事件处理方法中：

```
if(evt.getSource()==this.jFormattedTextFieldNumber){
    this.jLabelTextValue.setText(this.jFormattedTextFieldNumber.getText());
    Double aa=(Double)this.jFormattedTextFieldNumber.getValue();
```

```

        if(aa!=null){
            this.jLabelValueValue.setText(aa.toString());
        }
    }
}

```

(15) 将如下代码添加到 DemoJFormattedTextFieldFrame 的构造器内:

```
this.initFormatter();
```

(16) 最后将如下代码添加到 Demo 类的 public static void main(String[] args)方法中, 并删除原来方法体内的代码:

```

java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoJFormattedTextFieldFrame().setVisible(true);
    }
});

```

(17) 编译并运行主项目, 界面如图 4-19 所示。

(18) 在如图 4-19 所示的“数字格式”文本框中输入“123456789”后将光标移出“数字格式”文本框, 此时窗口如图 4-20 所示。

(19) 在“数字格式”文本框中输入含有非数字的字符“ABC123098”后将光标移出“数字格式”文本框, 此时窗口如图 4-21 所示。

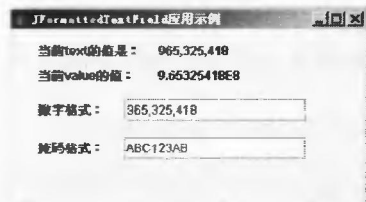


图 4-19 程序运行效果图

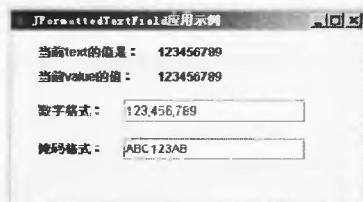


图 4-20 输入合法的数字

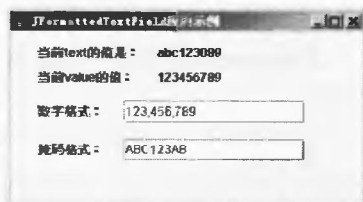


图 4-21 输入非法值

(20) 可以看到, 输入非法的值以后, text 的值会改变, 而 Value 的值仍为上一次的合法值, 并且对应控件也不会显示非法的输入值。

(21) 如果在“掩码格式”文本框中输入非法的值, 系统会发出报警声音, 并且不接受用户的输入。这里不再给出效果图, 读者可以自己试验这种情况。

4.6 Swing 文本区

JTextArea 是一种非常重要的文本输入控件。本节将对 JTextArea 展开讨论, 主要包括如下内容:

- JTextArea 简介;
- NetBeans 中 JTextArea 的使用。

4.6.1 JTextArea 类简介

JTextArea 是一种能够显示多行文本的文本区控件, 但是只能使用单一的字体和格式显示

文本。下面的代码显示了如何创建一个初始内容为“Hello! This is a JTextArea”的文本区。

```
JTextArea myTextArea=new JTextArea("Hello! This is a JTextArea");
```

JTextArea 类提供了 6 个构造器，表 4-19 列出了其中常用的 5 个。

表 4-19 JTextArea 类的常用构造器

构造器	说明
JTextArea()	构造一个新的 JTextArea，使用默认模型，初始字符串为 null，且行/列设置为 0
JTextArea(String text)	构造显示指定文本的新的 JTextArea，使用默认模型，行/列设置为 0
JTextArea(int rows, int columns)	构造具有指定行数和列数的新的空 JTextArea，使用默认模型，初始字符串为 null
JTextArea(String text,int rows, int columns)	构造具有指定文本、行数和列数的新的 JTextArea，使用默认模型
JTextArea(Document doc)	使用给定的文档模型构造一个新的 JTextArea，所有其他参数均默认值

JTextArea 本身不具有滚动窗格的功能，如果要实现该功能，需要将 JTextArea 添加到 JScrollPane 中，下面的代码显示了如何将上面创建的 myTextArea 添加到 JScrollPane 中，以使其具有滚动功能：

```
JScrollPane myScrollPane=new JScrollPane(myTextArea);
```

JTextArea 类提供了很多方法对 JTextArea 进行操作，表 4-20 列出了常用的一些方法及其说明。

表 4-20 JTextArea 类常用方法及说明

方法	说明
setLineWrap(boolean wrap)	设置文本区的换行策略，如果设置为 true，则当行的长度大于所分配的宽度时，将换行，如果设置为 false，则始终不换行，当策略更改时，将激发 PropertyChange 事件 ("lineWrap")，此属性默认为 false
getLineWrap()	获取文本区的换行策略
setWrapStyleWord(boolean word)	设置换行方式（如果文本区要换行），如果设置为 true，则当行的长度大于所分配的宽度时，将在单词边界（空白）处换行，如果设置为 false，则将在字符边界处换行，此属性默认为 false
getWrapStyleWord()	获取换行方式（如果文本区要换行）
getLineCount()	确定文本区中所包含的行数
getLineStartOffset(int line)	确定给定行起始处的偏移量
getLineEndOffset(int line)	确定给定行结尾处的偏移量
insert(String str,int pos)	将指定文本插入指定位置，如果模型为 null 或者文本为 null 或空，则不执行任何操作
append(String str)	将给定文本追加到文档结尾，如果模型为 null 或者字符串为 null 或空，则不执行任何操作
getRows()	获取 JTextArea 中的行数

续表

方 法	说 明
setRows(int rows)	设置此 JTextArea 的行数, 设置新值之后调用 invalidate()
getColumns()	获取 JTextArea 中的列数
setColumns(int columns)	设置此 JTextArea 中的列数, 设置新值之后执行 invalidate()

另外, JTextArea 还能使用其父类中提供的很多方法, 如使用 setFont(Font t)设置在 JTextArea 中显示文字的字体, 使用 setSize(int width)设置 JTextArea 的大小。需要用到这些方法的用户可以查阅相关的 API。

默认情况下, 文本区是可编辑的。可以使用 setEditable(false)将 JTextArea 设置为不可编辑。此时, JTextArea 仍然是可选择的, 用户也可以复制其中的数据, 但不能改变其中的数据。

JTextArea 使用 “\n” 作为内部表示的换行符号, 下面的代码显示了在 JTextArea 中添加一段文字后换行。

```
myTextArea.append("换行演示代码"+" \n" );
```

执行完上述代码后, 光标会自动定位到下一行的开始处。

提示 JTextArea 只是显示普通的文本, 不能对其进行格式化设置。若要显示格式化文本, 可以使用 JEditorPane, 在以后的章节中会对该类进行详细的介绍。

4.6.2 NetBeans 中 JTextArea 的使用

本小节将通过一个具体的例子讲述在 NetBeans 中使用 JTextArea 的方法。打开前面创建的项目 container, 按照如下步骤完成该例。

(1) 向 container 项目中添加一个通过 “JFrame Form” 模版创建的 DemoJTextAreaFrame 类。

(2) 添加完成以后, 打开 GUI 设计器, 在 “palette” 窗口中选择 “JTextArea” 选项, 之后在 GUI 设计器中的适当位置单击鼠标, 即可向 DemoJTextAreaFrame 类添加一个 JTextArea。

提示 将 JTextArea 添加到 GUI 设计器中的容器中时, NetBeans 会首先自动将该 JTextArea 添加到 JScrollPane 中, 使该 JTextArea 具有滚动功能, 然后再将 JScrollPane 添加到容器中。

(3) 用鼠标右键单击该 JTextField, 选择右键菜单中的 “Change Variable Name” 选项, 在弹出的对话框中将 JTextField 的名称修改为 jTextAreaTest。并在属性对话框中修改 jTextAreaTest 的属性, 将 lineWrap 和 wrapStyleWord 均设置为选中状态, 使 jTextAreaTest 在默认状态下可以换行。

(4) 接着向 DemoJTextAreaFrame 类中添加两个 JButton, 将这两个 JButton 的名字依次修改为 jButtonWrap、jButtonNoWrap, 并将其 text 属性的值分别修改为 “换行”、“不换行”。此时, GUI 设计器中的内容如图 4-22 所示。

(5) 分别为 jButtonWrap 和 jButtonNoWrap 按钮添加 ActionEvent 事件的处理方法。添加完成后, 源代码编辑器中会出现如图 4-23 所示的两个方法。

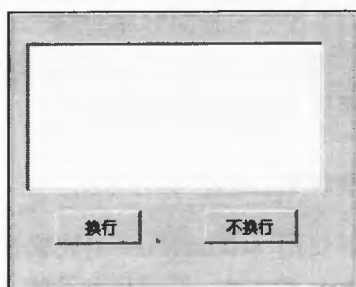


图 4-22 文本区的使用

```
private void jButtonNoWrapActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}  
  
private void jButtonWrapActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}
```

图 4-23 按钮事件处理方法

(6) `jButtonWrapActionPerformed(java.awt.event.ActionEvent evt)`方法用于处理“换行”按钮的 `ActionEvent` 事件，将如下代码添加到其方法中：

```
this.jTextAreaTest.setLineWrap(true);
```

提示 以上代码的作用是单击“换行”按钮后，`JTextAreaTest` 中内容能够自动换行。

(7) `jButtonNoWrapActionPerformed(java.awt.event.ActionEvent evt)`方法用来处理“不换行”按钮的 `ActionEvent` 事件，将如下代码添加到其方法中：

```
this.jTextAreaTest.setLineWrap(false);
```

提示 以上代码的作用是单击“不换行”按钮后，`JTextAreaTest` 中内容不能自动换行。

(8) 将如下代码添加到 `Demo` 类的 `public static void main(String[] args)`方法中，并删除原来方法体内的代码。

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoJTextAreaFrame().setVisible(true);  
    }  
});
```

(9) 编译并运行主程序，并输入一些内容，此时窗口中内容如图 4-24 所示。

(10) 单击“不换行”按钮，窗口内容如图 4-25 所示。此时如果单击换行按钮，窗口如图 4-24 所示。

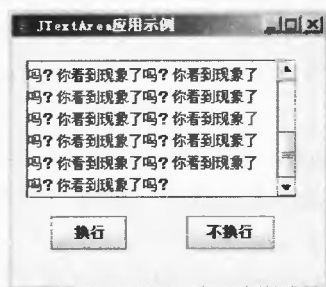


图 4-24 换行效果图

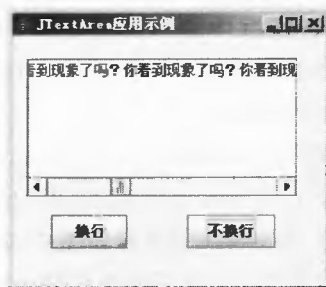


图 4-25 不换行效果图

4.7 单选按钮与复选框

在前面的章节已经为读者介绍了获取与设置用户输入的方法，有时可能会需要用户在已有的几种情况下做出选择，而不是要求用户输入。使用 `JCheckBox` 和 `JRadioButton` 可以完成这样的任务。本节具体介绍 `JCheckBox` 和 `JRadioButton`，主要包括如下内容：

- `JCheckBox` 类简介；
- `JRadioButton` 类简介；
- NetBeans 中 `JCheckBox` 与 `JRadioButton` 的使用。

4.7.1 JCheckBox 类简介

`JCheckBox` 扩展了 `javax.swing.JToggleButton`，也是一种常用的按钮。`JCheckBox` 主要包括如下特性。

- 有两个状态，选中与非选中，用户通过单击 `JCheckBox` 来选中该选项，再次单击就会取消选中。
- 允许用户从一组选项中进行多个选择。

以下代码为创建一个包含文本提示的 `JCheckBox`：

```
JCheckBox myJCheckBox=new JCheckBox("文本框");
```

以下代码为创建一个包含图标 `check.jpeg` 的 `JCheckBox`：

```
JCheckBox myJCheckBox=new JCheckBox("check.jpeg");
```

`JCheckBox` 通常被绘制成一个小方框，在选中选项时，这个方框中放置了一个“选中”标记。还可以为 `JCheckBox` 指定选中的图标，则该图标会取代选择框。此时，还应该再提供一个图标，用来区分选中状态与非选中状态。

`JCheckBox` 类一共有 7 个构造器，表 4-21 列出了其中常用的 5 个。

表 4-21 `JCheckBox` 类的常用构造器

构造器	说明
<code>JCheckBox(Icon icon)</code>	用指定的图标创建 <code>JCheckBox</code>
<code>JCheckBox(Icon icon, boolean selected)</code>	用指定的图标创建 <code>JCheckBox</code> ，并设置其默认状态
<code>JCheckBox(String text)</code>	用指定的文本创建 <code>JCheckBox</code>
<code>JCheckBox(String text, boolean selected)</code>	用指定的图标创建 <code>JCheckBox</code> ，并设置其默认状态
<code>JCheckBox(String text, Icon icon)</code>	用指定的文本和图标创建 <code>JCheckBox</code>

`JCheckBox` 有很多方法对各个属性进行操作，表 4-22 列出了常用方法及说明。

表 4-22 `JCheckBox` 类的常用方法及说明

方法	说明
<code>addItemListener(ItemListener l)</code>	向 <code>JCheckBox</code> 添加一个 <code>ItemListener</code>
<code>removeItemListener (ItemListener r)</code>	移除 <code>JCheckBox</code> 的 <code>ItemListener</code>

续表

方 法	说 明
isSelected()	返回 JCheckBox 的状态, 如果选定了 JCheckBox, 则返回 true, 否则返回 false
setSelected(boolean b)	设置 JCheckBox 的状态, 注意, 此方法不会触发 actionEvent 事件

另外 JCheckBox 可以使用其父类提供的很多方法, 如 setIcon()、setEnabled()、setDisabledIcon()等方法, 需要用到这些方法的用户可以查看相关的 API。

4.7.2 JRadioButton 类简介

JRadioButton 与 JCheckBox 一样也是 JToggleButton 的子类。JRadioButton 通常与其他的 JRadioButton 一起使用, 允许用户从一组 JRadioButton 中选择一个。JRadioButton 通常是成组使用的, 并且可以通过 ButtonGroup 管理。

ButtonGroup 允许按钮成组出现, 这样可以保证在任何时候都只有一个 JRadioButton 被选中。ButtonGroup 中通常包含 JRadioButton, 但这只是一种习惯, 可以使用 ButtonGroup 的 add()方法向其中添加任何类型的按钮。

ButtonGroup 是不可视的控件, 只提供了一组按钮的逻辑分组, 必须将 ButtonGroup 中的按钮添加到各自容器中才可以在界面上看到按钮控件。

以下代码显示了创建具有指定文本的 JRadioButton, 并将其添加到 ButtonGroup 中:

```
ButtonGroup rbGroup=new ButtonGroup();
JRadioButton one=new JRadioButton("示例按钮 1");
JRadioButton two=new JRadioButton("示例按钮 2");
rbGroup.add(one);
rbGroup.add(two);
```

提示 一般情况下, 添加到 ButtonGroup 中的按钮只有一个处于选中状态, 惟一例外的情况是初始状态中没有按钮被选中, 如果希望初始状态有某个按钮被选中, 需要进行设置。

JRadioButton 有 8 个构造器, 表 4-23 列出了常用的 5 个。

表 4-23 JRadioButton 常用构造器及其说明

构 造 器	说 明
JRadioButton(Icon icon)	创建一个初始化为未选择的单选按钮, 其有指定的图像但无文本
JRadioButton(Icon icon,boolean selected)	创建一个具有指定图像和选择状态的单选按钮, 无文本
JRadioButton(String text)	创建一个具有指定文本的状态为未选择的单选按钮
JRadioButton(String text , boolean selected)	创建一个具有指定文本和选择状态的单选按钮
JRadioButton(String text,Icon icon)	创建一个具有指定文本和图像, 并初始化为未选择的单选按钮

JRadioButton 继承了其父类的所有属性和大部分的默认值, 但是默认情况下 JRadioButton 没有绘制边框, 并且其 horizontalAlignment 被设置为 LEADING。如果用户需要用到其父类中的方法, 可以查阅 API, 这里不再赘述。

JRadioButton 没有定义新的事件，其使用从父类继承过来的事件处理方法。如使用 `actionPerformed(ActionEvent e)` 处理 `ActionEvent` 事件，使用 `itemStateChanged(ItemEvent e)` 处理 `ItemEvent` 事件，使用 `stateChanged(ChangeEvent e)` 处理 `ChangeEvent` 事件。

4.7.3 开发使用 JCheckBox 和 JRadioButton 的界面

在 NetBeans 中，通过鼠标拖曳，即可创建 JCheckBox 和 JRadioButton，为开发人员提供了很大的方便。本小节将通过一个例子讲述在 NetBeans 中使用 JCheckBox 和 JRadioButton 的方法。打开前面开发的项目 `container`，按如下步骤完成该例。

(1) 向 `container` 项目中添加一个 JFrame Form 模版创建的 `DemoJCheckRadioFrame` 类。

(2) 添加完成以后，打开 GUI 设计器，在“Palette”窗口中选择“ButtonGroup”选项，之后在 GUI 设计器中的适当位置单击鼠标，即可向 `DemoJCheckRadioFrame` 类添加一个 `ButtonGroup`。

(3) 用鼠标右键单击新添加的 `ButtonGroup`，选择右键菜单中的“Change Variable Name”选项，在弹出的对话框中将 `ButtonGroup` 的名称修改为 `buttonGroupOne`。

(4) 在“palette”窗口中选择“JCheckbox”选项，之后在 GUI 设计器中的适当位置单击鼠标，即可向 `DemoJCheckRadioFrame` 类添加一个 `JCheckBox`。

(5) 在“palette”窗口中选择“JRadioButton”选项，在 GUI 设计器中的适当位置单击鼠标，即可向 `DemoJCheckRadioFrame` 类添加一个 `JRadioButton`。

(6) 用鼠标右键单击 `JCheckBox`，选择右键菜单中的“Change Variable Name”选项，在弹出的对话框中将该 `JCheckBox` 的名称修改为 `jCheckBoxBold`。按照同样的方法，将 `JRadioButton` 的名称修改为 `jRadioButtonSmall`。

(7) 修改 `jCheckBoxBold` 和 `jRadioButtonSmall` 的 `text` 属性，分别将其设置为“黑体”、“缩小”。并将 `jRadioButtonSmall` 的 `buttonGroup` 属性的值设置为 `buttonGroupOne`。

(8) 按照上述方法，再向 `DemoJCheckRadioFrame` 添加一个 `JCheckBox` 和两个 `JRadioButton`，并按照如表 4-24 所示修改其属性。

表 4-24 控件属性与值对照表

控 件	名 称	text 属性值	是 否 选 中
JCheckBox	JCheckBoxItalic	斜体	否
JRadioButton	JRadioButtonCommon	普通	是
JRadioButton	JRadioButtonbBig	加大	否

(9) 将 `JRadioButtonCommon` 与 `JRadioButtonbBig` 的 `buttonGroup` 的属性设置为 `buttonGroupOne`。

(10) 向 `DemoJCheckRadioFrame` 中添加一个 `JLabel`，用来显示信息。将其名称修改为 `jLabelMessage`，`text` 属性的值修改为 `JCheckBox` 与 `JRadioButton`，并将其 `horizontalAlignment` 属性的值设置为“CENTER”。此时 GUI 设计器中的内容如图 4-26 所示。

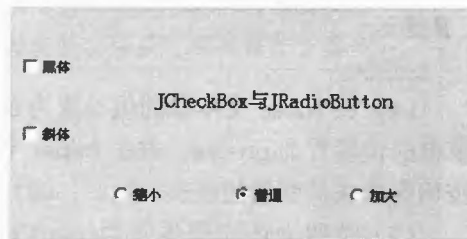


图 4-26 GUI 设计器中的内容

此时已经成功地完成了界面的开发。

4.7.4 开发业务代码

完成了界面的开发后, 本节将开发此程序的业务代码。按照如下步骤完成业务代码的开发。

(1) 选中“黑体”复选框, 单击鼠标右键, 在弹出的菜单中依次选择“Events”/“Item”/“itemStateChanged”选项, 为“黑体”复选框添加 ItemEvent 事件的处理方法。

(2) 按照上述步骤, 依次为 JRadioButtonSmall、JCheckBoxItalic、JRadioButtonCommon 和 JRadioButtonbBig 添加 ItemEvent 事件的处理方法。此时, 源代码编辑器中会出现如图 4-27 所示的方法。



提示

图 4-27 所示的 5 个方法分别用来处理 JCheckBoxBold、JCheckBoxItalic、JRadioButtonSmall、JRadioButtonCommon 以及 JRadioButtonBig 的 ItemEvent 事件。

(3) 向 DemoJCheckRadioFrame 添加一个成员变量。在属性窗口中依次选中“Source Packages”/“org.netbeans.swing.container”/“DemoJCheckRadioFrame.java”/“DemoJCheckRadioFrame”/“Fields”节点, 单击鼠标右键, 在弹出的菜单中选择“Add Field”选项, 此时弹出如图 4-28 所示的界面。

```
private void JRadioButtonBigItemStateChanged(java.awt.event.ItemEvent evt) {
    // TODO add your handling code here
}

private void JRadioButtonCommonItemStateChanged(java.awt.event.ItemEvent evt) {
    // TODO add your handling code here
}

private void JRadioButtonSmallItemStateChanged(java.awt.event.ItemEvent evt) {
    // TODO add your handling code here
}

private void JCheckBoxItalicItemStateChanged(java.awt.event.ItemEvent evt) {
    // TODO add your handling code here
}

private void JCheckBoxBoldItemStateChanged(java.awt.event.ItemEvent evt) {
    // TODO add your handling code here
}
```

图 4-27 源代码编辑器中方法

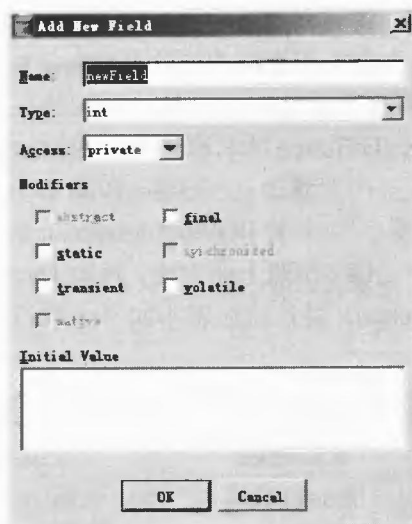


图 4-28 添加成员变量对话框



提示

可以在如图 4-28 所示的窗口中设置要添加变量的名称、类型、访问限制修饰符以及是否为最终的, 是否为静态的以及成员变量的初始值等内容。

(4) 将 Name 文本框的值设置为 style, Type 文本框的值设置为 int, 并将 Access 下拉列表框的值设置为 private。并在 Initial Value 文本区中输入 java.awt.Font.PLAIN, 单击“OK”按钮完成成员变量的添加。此时, 源代码编辑器中出现如图 4-29 所示的内容。

(5) 按照上述步骤再向 DemoJCheckRadioFrame 添加另一个成员变量, 名称为“font-Size”, 类型为 int, 访问限制修饰符为 private, 初始值为 18。此时, 源代码编辑器中出现如

图 4-30 所示的内容。

```
private int style = java.awt.Font.PLAIN
```

图 4-29 添加 style 成员变量

```
private int fontSize = 18;
```

图 4-30 添加 fontSize 成员变量

(6) 向 DemoJCheckRadioFrame 添加一个方法。在属性窗口中依次选中 DemoJCheckRadioFrame 下的 Methods 节点，单击鼠标右键，在弹出的菜单中选择“Add Method”选项，此时弹出如图 4-31 所示的界面。

可以在如图 4-31 所示的窗口中设置要添加方法的名称，返回值类型，访问限制修饰符以及是否为最终的、抽象的、静态的、同步的、本地的，还可以设置方法的参数以及异常等内容。

(7) 将 Name 的值设置为 setStyle，Return Type 的值设置为 void，并将 Access 下拉列表框的值设置为 private。单击“Add”按钮，弹出如图 4-32 所示的对话框。

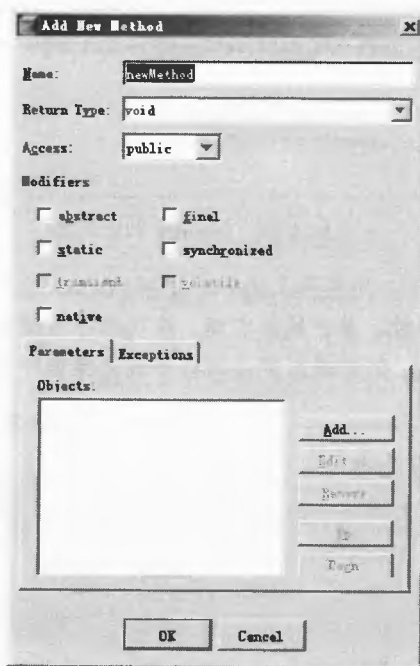


图 4-31 添加方法对话框

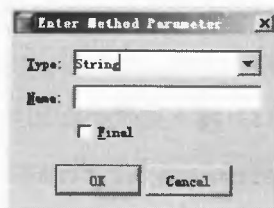


图 4-32 添加方法参数对话框

(8) 在如图 4-32 所示对话框中设置参数的类型和名称，将 Type 的值设置为 int，将 Name 的值设置为 style。单击“OK”按钮完成设置，返回如图 4-31 所示的界面，发现“Parameters”列表中多出了一个参数，如图 4-33 所示。

(9) 按照上述步骤再向 DemoJCheckRadioFrame 中添加一个方法，方法名称为“getStyle”，返回值为 int，访问限制修饰符为 private，没有入口参数。

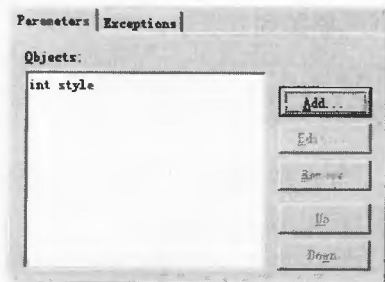


图 4-33 添加参数后的“Parameters”列表

(10) 添加完成以后, 源代码编辑器中出现如图 4-34 所示的两个方法。

(11) 将如下代码添加到 `setStyle(int style)` 方法的方法中:

```
this.style=this.style+style;
```

(12) 将如下代码添加到 `getStyle()` 方法的方法中:

```
return this.style;
```

以上两个方法在 Java 中被称为 `style` 属性的访问器, 按照 Java 的编程规范, 所有的属性都应该有与之对应的 `get` 和 `set` 访问器, 对任何属性的修改和访问都应该通过访问器, `get` 访问器用来获取属性值, `set` 访问器用来设置属性值, 这是一种非常好的编程习惯, 可以很好地体现面向对象中的封装性。访问器方法的命名规则是 `get` 或 `set` 加上属性名称, 入口参数类型与对应的属性类型相同。

(13) 按照上述步骤为 `DemoJCheckRadioFram` 的另一个成员变量 `fontSize` 添加访问器。添加完成以后, 源代码编辑器中出现如图 4-35 所示的方法。

```
private void setStyle(int style) {  
}  
  
private int getStyle() {  
}
```

图 4-34 style 的 `get` 与 `set` 方法

```
private void setFontSize(int size) {  
}  
  
private int getFontSize() {  
}
```

图 4-35 `fontSize` 的访问器

为一个属性编写 `get` 和 `set` 方法对其进行封装不一定要手工进行, 用 NetBeans 也可以快速地实现。具体做法是选中要进行封装的属性, 单击鼠标右键, 依次选择右键菜单中的“Refactor”/“Encapsulate Fields...”选项, 然后根据弹出的向导进行操作即可自动生成对应属性的 `get` 和 `set` 访问器方法。

(14) 将下列代码添加到 `setFontSize(int fontSize)` 方法中:

```
this.fontSize = fontSize;
```

(15) 将下列代码添加到 `getFontSize()` 方法中:

```
return this.fontSize;
```

(16) 向 `DemoJCheckRadioFram` 中添加名为“setLableFont”, 返回值为 `void`, 访问限制修饰符为 `private` 的方法。该方法的作用是设置 `JLabelMessage` 上显示内容的字体格式。将如下代码添加到方法中:

```
this.jLabelMessage.setFont(new java.awt.Font("宋体",this.getStyle(),this.  
getFontSize()));
```

(17) 向 `JRadioButtonSmallItemStateChanged` 方法内添加如下代码:

```
this.setFontSize(14);  
this.setLableFont();
```

(18) 向 `JRadioButtonCommonItemStateChanged` 方法内添加如下代码:

```
this.setFontSize(18);  
this.setLableFont();
```

(19) 向 `jButtonBigItemStateChanged` 方法内添加如下代码:

```
this.setFontSize(22);  
this.setLableFont();
```

- 以上 3 个方法代码的作用是根据不同的选项设置 `fontSize` 属性的值。
- 调用 `setLableFont()` 方法设置 `JLabelMessage` 标签上显示内容的字体。

(20) 向 `jButtonItalicItemStateChanged` 方法内添加如下代码:

```
if(this.jCheckBoxItalic.isSelected()){  
    this.setStyle(java.awt.Font.ITALIC);  
}else {  
    this.setStyle(-java.awt.Font.ITALIC);  
} this.setLableFont();
```

(21) 向 `jButtonBoldItemStateChanged` 方法内添加如下代码:

```
if(this.jCheckBoxBold.isSelected()){  
    this.setStyle(java.awt.Font.BOLD);  
}else{  
    this.setStyle(-java.awt.Font.BOLD);  
} this.setLableFont();
```

- 以上两段代码的作用是根据复选框的选中状态来设置 `style` 属性的值。
- 调用 `setLableFont()` 方法以设置 `JLabelMessage` 标签上显示内容的字体。

(22) 将如下代码添加到 `Demo` 类的 `public static void main(String[] args)` 方法中,并删除原来方法中的代码:

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoJCheckRadioFram().setVisible(true);  
    }  
});
```

至此,已经成功地完成了代码的开发。

4.7.5 程序功能演示

经过前面操作,现在已经成功地开发了该程序。本节将对该程序的功能进行演示。编译并运行主程序,界面如图 4-36 所示。单击“加大”单选按钮以后,程序运行界面如图 4-37 所示。

然后单击“黑体”复选框,程序运行界面如图 4-38 所示。再单击“斜体”复选框,程序运行界面如图 4-39 所示。

单击不同的按钮,还会看到本程序运行时的其他几种效果,这里就不一一给出了。有兴趣的读者可以试验其他几种情况。



图 4-36 程序运行界面



图 4-37 “加大”界面



图 4-38 “加大”、“黑体”界面



图 4-39 “加大”、“黑体”、“斜体”界面

4.8 编辑器面板

JEditorPane 是一种功能非常强大的文本编辑与显示控件。本节将对 JEditorPane 展开讨论，主要包括如下内容：

- JEditorPane 类简介；
- NetBeans 中 JEditorPane 的使用。

4.8.1 JEditorPane 类简介

JEditorPane 是 JTextComponent 的子类，其可以显示多种格式的内容，如 HTML 和 RTF，可以用做浏览简单 HTML 的工具。

JEditorPane 是 Swing 的风格化文本控件的基础，能够显示图文混排的效果，同时还提供了对自定义文件格式的支持。下面的代码为创建一个最简单的 JEditorPane：

```
JEditorPane myJEditorPane=new JEditorPane();
```

JEditorPane 有 4 个构造器，表 4-25 列出了这些构造器并做了相应的说明。

表 4-25 JEditorPane 构造器及其说明

构造器	说明
JEditorPane()	创建一个不带参数的 JEditorPane
JEditorPane(java.net.URL initialPage)	根据用作输入的指定 URL 创建一个 JEditorPane
JEditorPane(String url)	根据包含 URL 规范的字符串创建一个 JEditorPane
JEditorPane(String type,String text)	创建一个已初始化为给定文本内容的 JEditorPane，这是同时调用 setContentType 和 setText 方法的便捷构造方法

JEditorPane 提供了很多用于对 JEditorPane 进行操作的方法, 表 4-26 列出了其中常用的方法并做了相应的说明。

表 4-26

JEditorPane 常用方法及说明

方 法	说 明
addHyperlinkListener(HyperlinkListener listener)	添加一个超链接侦听器
removeHyperlinkListener(HyperlinkListener listener)	移除超链接侦听器
getHyperlinkListeners()	获取使用 addHyperlinkListener() 添加到 JEditorPane 所有 HyperLinkListener 的数组, 如果尚未添加侦听器, 则返回一个空数组
getPage()	得到当前正在显示的 URL, 如果在文档创建中未指定 URL, 则此方法返回 null, 并且不解析相关的 URL
setPage(String url)	设置当前要显示的 URL, 参数为 String 类型
setPage(java.net.URL page)	设置当前要显示的 URL, 参数为 URL 类型
read(InputStream in, Object desc)	此方法根据流进行初始化
getStream(java.net.URL page)	获取给定 URL 的流, 该 URL 是将来由 setPage 方法加载的, 默认情况下, 这只是简单地打开该 URL 并返回流。可重新实现此方法以执行有用的任务, 例如从缓存中获取流、监视该流的进程等
getContentType()	获得设置此编辑器当前要处理的内容类型, 将其定义为与当前已安装的 EditorKit 相关联的类型
setContentType(String type)	设置此编辑器所处理的内容类型
replaceSelection()	用由给定字符串表示的新内容替换当前选中的内容, 如果没有选中的内容, 那么此操作就等效于插入给定文本, 如果没有替换文本 (也就是 content 字符串为空或为 null), 那么就等效于移除当前的选择。替换文本具有当前为输入所定义的各种属性。如果该组件是不可编辑的, 则发出蜂鸣声并返回
setText(String t)	将此 TextComponent 的文本设置为指定内容, 以此编辑器的内容类型格式提供该内容。例如, 如果将类型设置为 text/html, 则应按照 HTML 指定该字符串
getText()	根据此编辑器的内容类型返回 TextComponent 所包含的文本, 如果试图检索文本的同时抛出异常, 则返回 null

JEditorPane 还能够使用从父类继承来的很多方法, 这里不再列举, 如果用户需要, 可以查阅相关的 API。

JEditorPane 可以在用户单击超级链接时触发一个 HyperlinkEvent 的事件。程序通常通过加载一个新页面来响应该事件。通过添加一个 HyperlinkListener 监听器来监听 HyperlinkEvent 事件。

HyperlinkListener 监听接口中只有一个 hyperlinkUpdate(HyperlinkEvent e) 方法, 当 JEditorPane 中发生 HyperlinkEvent 事件时, 即可调用该方法处理 HyperlinkEvent 事件。

目前 JEditorPane 的功能还很有限, 用其制作的 HTML 阅读器只能够显示简单的 HTML 文件, 但是无法显示通常在 Web 上找到的复杂页面。JEditorPane 最常见的应用是制作 HTML 格式的帮助文档。



4.8.2 NetBeans 中 JEditorPane 的使用

通过使用 NetBeans，可以使得创建 JEditorPane 非常简单、快捷。本节将通过一个例子，介绍在 NetBeans 中使用 JEditorPane 的方法。打开前面创建的项目 container，按照如下步骤完成该例。

(1) 向 container 项目中添加一个通过 JFrame Form 模版创建的 DemoJEditorPane-Frame 类。

(2) 添加完成以后，打开 GUI 设计器，在“Palette”窗口中选择“JEditorPane”选项，之后在 GUI 设计器中的适当位置单击鼠标，即可向 DemoJEditorPaneFrame 中添加一个 JEditorPane。

JEditorPane 本身不具备自动滚动功能，需要将其放入 JScrollPane 才能实现滚动。在
提示 NetBeans 中使用 JEditorPane 时，NetBeans 会自动将 JEditorPane 放入 JScrollPane 中，然后再将其添加到容器中。

(3) 用鼠标右键单击 JEditorPane，选择右键菜单中的“Change Variable Name”选项，在弹出的对话框中将 JEditorPane 的名称修改为 jEditorPaneHTML。并在属性对话框中修改 jEditorPaneHTML 的属性，将 editable 属性的值设置为非选中状态。

(4) 依次向 DemoJEditorPaneFrame 中添加 JLabel、JTextField、JButton 控件，并按照如表 4-27 所示修改其相应属性。

表 4-27

修改属性对照表

控 件 类 型	名 称	text 属性
JLabel	jLabelSite	地址
JTextField	jTextFieldAddress	空
JButton	jButtonGo	转到

(5) 此时，GUI 设计器中内容如图 4-40 所示。

(6) 选中 jEditorPaneHTML，单击鼠标右键，在弹出的菜单中依次选择“Events”/“HyperLink”/“HyperlinkUpdate”选项，为 jEditorPaneHTML 添加 HyperLinkEvent 事件的处理方法。此时，源代码编辑器中出现如图 4-41 所示内容。

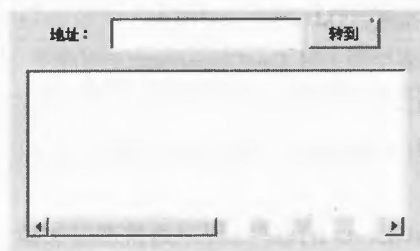


图 4-40 GUI 设计器中内容

```
private void jEditorPaneHTMLHyperlinkUpdate(javax.swing.event.HyperlinkEvent evt) {  
    TODO add your handling code here.  
}
```

图 4-41 jEditorPaneHTML 的 HyperLinkEvent 事件处理方法

(7) 将下列代码添加到 jEditorPaneHTMLHyperlinkUpdate 方法的方法中，该方法用于处理 jEditorPaneHTML 的 HyperLinkEvent 事件。

```
try{  
    if(evt.getEventType()==javax.swing.event.HyperlinkEvent.EventType.  
    ACTIVATED)
```

```

jEditorPaneHTML.setPage(evt.getURL());
} catch (java.io.IOException e) {
    e.printStackTrace();
}
}

```

(8) 为 jButtonGo 按钮添加 ActionEvent 的事件处理方法 jButtonGoActionPerformed, 并将如下代码添加到该方法的方法中:

```

String myURL=this.jTextFieldAddress.getText().trim();
try{
    if(myURL!=null&&!myURL.equals(""))
        jEditorPaneHTML.setPage(myURL);
} catch (java.io.IOException e) {
    e.printStackTrace();
}

```

(9) 将下列代码添加到 Demo 类的 public static void main(String[] args)方法中, 并删除原来方法中的代码:

```

java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoJEditorPaneFrame().setVisible(true);
    }
});

```

(10) 编译并运行主项目, 在 jTextFieldAddress 中输入 http://www.163.com, 单击“转到”按钮, 此时窗口内容如图 4-42 所示。单击“体育”链接, 出现如图 4-43 所示的界面。

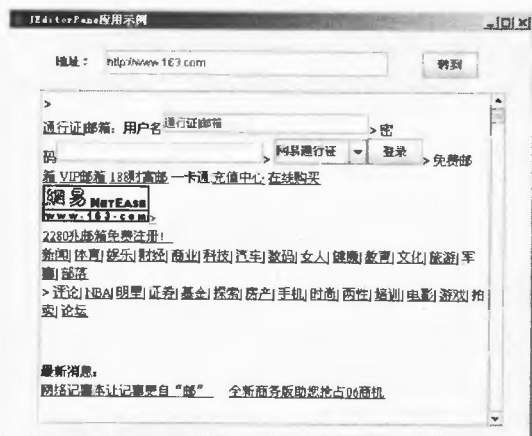


图 4-42 程序运行效果图



图 4-43 单击“体育”后效果图

提示

如果不将 jEditorPaneHTML 的 editable 属性的值设置为非选中状态, 则单击页面上的超级链接时不会产生 HyperLinkEvent 事件。在实际应用中, 开发人员可能需要在页面加载时变换光标或对异常进行处理, 该例只是在 JEditorPane 中使用超级链接的简单方法。

从以上两图可以看出，JEditorPane 并不能很好地显示 Web 上找到的复杂的页面。不过其功能也在不断地完善，相信有一天 JEditorPane 能够很好地显示 Web 上找到的复杂页面。

4.9 小结

本章主要介绍了 Swing 的几种常用控件，并介绍了如何在 NetBeans 中使用这些控件进行项目开发。使用 Swing 提供的可视化开发环境，可以大大地缩短 Swing 项目的开发周期，提高生产效率。在下一章中，将会对另外几种常用控件进行介绍。

第 5 章

Swing 常用高级控件

上一章对 Swing 中常用的几种基本控件进行了详细介绍。本节将介绍其他几种常用的高级 Swing 控件，主要包括如下内容：

- Swing 列表；
- Swing 组合框；
- Swing 分割窗口；
- 使用 NetBeans 构建 Swing 的综合例程。

5.1 Swing 列表

如果需要向用户显示一组选项，而单选按钮或复选框的设置需要较大的空间，则可以使用 JList 控件。本节将对 JList 进行介绍，主要包括如下内容：

- JList 类简介；
- NetBeans 中 JList 的使用。

5.1.1 JList 类简介

JList 控件类似于一组复选框或一组单选按钮，JList 的各个项目是放在单个列表框中，它是通过单击选项本身而不是单击按钮来选定的。可以通过设置，允许对列表中的项目进行多项选择，这样用户就可以选定列表中项目的任意组合。

以下代码创建了一个包含 4 个选项的 JList：

```
String [] fruit={"apple","pear","orange","banana"}  
JList myJList=new JList(fruit);
```

JList 不支持自动滚动的功能，若要实现该功能，需要将 JList 添加到 JScrollPane 中。将 JList 添加到 JScrollPane 中的代码如下：

```
JScrollPane myJScrollPane=new JScrollPane();
```

```
myJScrollPane.add(myJList);
```

JList 类提供了 4 个构造器，如表 5-1 所示。

表 5-1 JList 类的构造器及其简单说明

构造器	说 明
JList()	构造一个使用空模型的 JList
JList(Object[] listData)	构造一个 JList，使其显示指定数组中的元素，此构造方法仅委派给 ListModel 构造方法
JList(ListModel dataModel)	构造一个 JList，使其使用指定的非 null 模型显示元素，所有 JList 构造方法都委派给此方法
JList(Vector listData)	构造一个 JList，使其显示指定 Vector 中的元素，此构造方法仅委派给 ListModel 构造方法

JList 提供了大量的方法进行相应的操作，具体方法如表 5-2 所示。

表 5-2 JList 的常用方法及说明

方 法	说 明
getSelectionForeground()	获取选择的前景色
setSelectionForeground(Color selectionForeground)	设置所选单元的前景色，单元渲染器可以使用此颜色呈现所选单元的文本和图形
getSelectionBackground()	获取所选单元的背景色
setSelectionBackground(Color selectionBackground)	设置所选单元的背景色，单元渲染器可以使用此颜色填充所选单元，此属性的默认值由外观实现定义
setVisibleRowCount(int visibleRowCount)	设置不使用滚动条可以在列表中显示的首选行数
getLayoutOrientation()	如果布局是单列单元，则返回 JList.VERTICAL；如果布局是“报纸样式”并且内容按先纵向后横向流动，则返回 JList.VERTICAL_WRAP；如果布局是“报纸样式”并且内容按先横向后纵向流动，则返回 JList.HORIZONTAL_WRAP
setLayoutOrientation(int layoutOrientation)	定义布置列表单元的方式，可以采用的值为 JList.VERTICAL，JList.HORIZONTAL_WRAP，JList.VERTICAL_WRAP
setDragEnabled(boolean b)	设置 dragEnabled 属性，该属性必须为 true，才能启用对此组件的自动拖动处理，启用自动拖动处理时，只要用户在选择上按下鼠标按键，然后将鼠标移动几个像素，大多数外观就开始拖放操作了。因此，将此属性设置为 true 可以对选择的行为方式产生微妙的影响
getDragEnabled()	获取 dragEnabled 属性
getModel()	获取保存由 JList 组件显示的项列表的数据模型
setModel(ListModel model)	设置表示列表内容或“值”的模型，并在通知 PropertyChangeListener 之后清除列表选择
setListData(Object[] listData)	根据一个 Object 数组构造 ListModel，然后对其应用 setModel
setListData(Vector listData)	根据 Vector 构造 ListModel，然后对其应用 setModel

续表

方 法	说 明
<code>addListSelectionListener(ListSelectionListener listener)</code>	为每次选择发生更改时要通知的列表添加侦听器
<code>removeListSelectionListener(ListSelectionListener listener)</code>	从每次选择发生更改时要通知的列表中移除侦听器
<code>getListSelectionListeners()</code>	获取使用 <code>addListSelectionListener()</code> 添加到 <code>JList</code> 中的所有 <code>ListSelectionListener</code> 组成的数组
<code>setSelectionModel(ListSelectionModel selectionModel)</code>	将列表的 <code>selectionModel</code> 设置为非 <code>null</code> 的 <code>ListSelectionModel</code> 实现。选择模型可以完成单个选择、连续范围选择和非连续范围选择等任务
<code>setSelectionMode(int selectionMode)</code>	确定允许单项选择还是多项选择
<code>getSelectionMode()</code>	获取允许单项选择还是多项选择
<code>getMinSelectionIndex()</code>	获取选择的最小单元索引
<code>getMaxSelectionIndex()</code>	获取选择的最大单元索引
<code>isSelectedIndex(int index)</code>	判断当前选中的索引是否是指定的索引
<code>isSelectionEmpty()</code>	如果没有选择, 则返回 <code>true</code>
<code>clearSelection()</code>	清除选择, 调用此方法后, <code>isSelectionEmpty</code> 将返回 <code>true</code>
<code>setValuesAdjusting(boolean b)</code>	将数据模型的 <code>isAdjusting</code> 属性设置为 <code>true</code> , 这样完成所有选择事件时就会生成单个事件 (例如, 在选择模式的列表上拖动鼠标时)
<code>getValuesAdjusting()</code>	获取数据模型的 <code>isAdjusting</code> 属性的值, 如果进行了多处更改, 则此值为 <code>true</code>
<code>getSelectedValues()</code>	获取所选单元的一组值, 返回值以递增的索引顺序存储
<code>getSelectedIndex()</code>	获取所选的第一个索引; 如果没有选择项, 则返回 <code>-1</code>
<code>getSelectedValue()</code>	所选的第一个值, 如果选择为空, 则返回 <code>null</code>
<code>setSelectedValue(Object anObject, boolean shouldScroll)</code>	从列表中选择指定的对象

表 5-2 所列的 `setSelectionMode(int selectionMode)` 方法用于设置对 `JList` 中的选项进行单项选择还是多项选择。`selectionMode` 值有以下几种。

- `ListSelectionModel.SINGLE_SELECTION` 一次只能选择一个列表索引。
- `ListSelectionModel.SINGLE_INTERVAL_SELECTION` 一次可以选择一个连续的索引间隔。
- `ListSelectionModel.MULTIPLE_INTERVAL_SELECTION` 在此模式中, 不存在对选择的限制。这是默认设置。

5.1.2 控件 MVC 思想简介

在使用 `JList` 等控件进行编程的时候, 需要用到 MVC 的思想, 很好地理解和掌握这种思想, 可以使开发人员在开发中有事半功倍的效果, 下面就对 MVC 思想做简单的介绍。

Swing 使用模型—视图—控制器体系结构 (Model-View-Controller MVC) 作为其每个控

件的基本设计思想, MVC 将 GUI 控件拆分为 3 个元素, 每个元素都起着至关重要的作用。各个元素的作用如下。

- 模型 (Model): 包括每个控件的状态数据, 不同控件有不同的模型。不论控件在屏幕上是如何被描绘的, 这些信息总是相同的, 也就是说模型数据与控件的可视化表示是相互独立的。
- 视图 (View): 视图是控件在屏幕上的表现形式。
- 控制器 (Controller): 控制器是用户界面的一部分, 其指示控件如何与事件进行交互。事件的形式有多种, 如鼠标单击、获得或者失去焦点、键盘敲击等。控制器决定了每个控件如何对不同的事件做出相应的反应。

Swing 通常使用一个叫做模型代理 MVC 设计的简单变体, 该设计将视图和控制器对象合并到一个元素中, 即 UI 代理。该元素将控件绘制到屏幕上, 并处理 GUI 事件。

在 MVC 中将模型与 UI 代理进行区分, 可以将多个视图与一个模型绑在一起, 如果需要更改数据, 只需在一个地方就可以完成更改操作。

从表 5-1 中可以看出, JList 可以使用很多种模型, 常见的如数组对象以及向量等。使用相应的模型构造 JList, 则可以很方便地管理 JList 中的数据。这样开发人员不需要专门学习每种不同控件对数据的管理, 只要学会使用向量、数组等即可, 非常方便高效。

5.1.3 NetBeans 中开发使用 JList 程序

在 NetBeans 中只需要用鼠标拖曳即可创建 JList。本节将通过一个例子展示在 NetBeans 中使用 JList 的方法。打开上一章中创建的项目 container, 按如下步骤完成该例。

(1) 向其中添加一个使用 JFrame Form 模版创建的 DemoJListFrame 类。

(2) 添加完成以后, 打开 GUI 设计器, 在 “palette” 窗口中选择 “JList” 选项, 之后在 GUI 设计器中的适当位置单击鼠标, 即可向 DemoJListFrame 中添加一个 JList。

(3) 用鼠标右键单击该 JList, 选择右键菜单中的 “Change Variable Name” 选项, 在弹出的对话框中将该 JList 的名称修改为 jListCountryList。

(4) 在属性对话框中, 单击 model 属性右侧的  按钮, 弹出如图 5-1 所示的对话框。

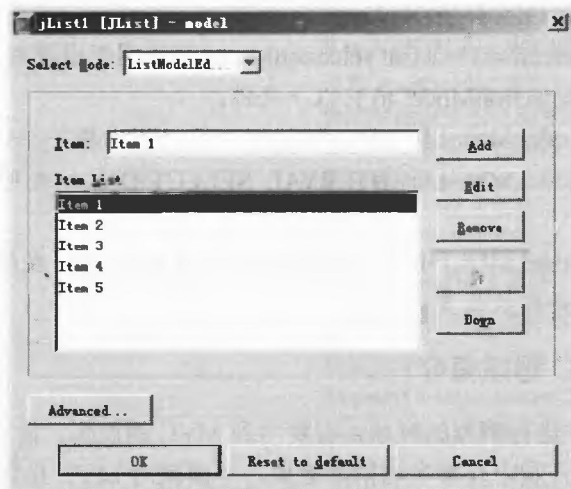


图 5-1 设置 model 属性对话框

(5) 选中图 5-1 中“Item List”列表框的“Item 1”选项，单击“Remove”按钮将其删除。重复该操作，清空“Item List”列表框中的所有选项，单击“OK”按钮完成。

(6) 选中 `JListCountryList`，单击鼠标右键，在弹出的菜单中依次选择“Events”/“ListSelection”/“valueChanged”选项，为 `JListCountryList` 添加 `ListSelection` 事件的处理方法。此时，源代码编辑器中出现如图 5-2 所示的内容。

```
private void JListCountryListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    // TODO add your handling code here
}
```

图 5-2 `JListCountry` 的 `ListSelection` 事件处理方法

(7) 依次向 `DemoJListFrame` 类添加两个 `JLabel`，两个 `JButton` 和一个 `JTextField`，并按照如表 5-3 所示修改其属性。

表 5-3

修改属性对照表

控件类型	名称	text 属性
<code>JLabel</code>	<code>jLabelList</code>	国家列表
<code>JLabel</code>	<code>jLabelActionMessage</code>	你还没有进行任何操作
<code>JButton</code>	<code>jButtonAdd</code>	添加
<code>JButton</code>	<code>jButtonDelete</code>	删除
<code>JTextField</code>	<code>jTextFieldCountryName</code>	空

此时，GUI 设计器中的内容如图 5-3 所示。

(8) 分别为 `jButtonAdd` 和 `jButtonDelete` 添加 `ActionEvent` 事件处理方法。

(9) 向 `DemoJListFrame` 类添加一个成员变量，将其名称修改为 `countryVector`，类型为 `Vector`，访问限制修饰符为 `private`。此时源代码编辑器中会提示没有导入 `java.util.Vector`，按提示导入该类。

(10) 将下列代码添加到 `JListCountryListValueChanged` 的方法中。此方法用来处理 `JListCountryList` 的 `ListSelection` 事件，根据用户在列表中选择值，确定在 `jLabelActionMessage` 上显示的信息。

```
String choosedCountry=(String)this.jListCountryList.getSelectedValue();
this.jLabelActionMessage.setText("你从列表中选择"+choosedCountry+"选项");
```

(11) 将下列代码添加到 `JButtonAddActionPerformed` 方法中，此方法用来处理 `JButtonAdd` 的 `ActionEvent` 事件。

```
String countryName=this.jTextFieldCountryName.getText().trim();
if(countryName!=null&&!countryName.equals("")){
    this.imageVector.add(countryName);//将新增选项添加到向量模型中
    this.jLabelActionMessage.setText("你将"+countryName+"添加到列表中");
    this.jListCountryList.setListData(this.imageVector);//为 JList 刷新数据模型
}else{
```

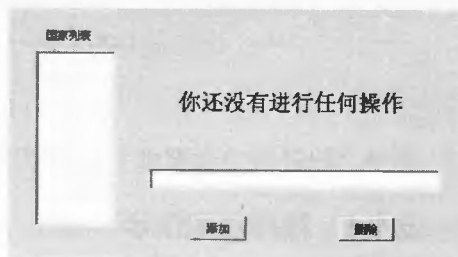


图 5-3 GUI 设计器中内容

imageVector ??
or countryVector

```
this.jLabelActionMessage.setText("请先输入内容再添加");  
}
```

- jButtonAddActionPerformed 方法的作用是：如果在 jTextFieldCountryName 中输入的内容不为空，则将输入的内容添加到 jListCountryList 中。
- 从代码中可以看到对 JList 中数据的管理是通过向量 imageVector 来完成的，这样非常方便，同时也体现了 MVC 的思想。

(12) 将下列代码添加到 jButtonDeleteActionPerformed 方法中，此方法用来处理 jButtonDelete 的(ActionEvent) 事件，作用是把选中的选项从列表中删除。

```
String countryName=(String)this.jListCountryList.getSelectedValue();  
if(countryName!=null){  
    this.imageVector.remove(countryName);  
    this.jListCountryList.setListData(imageVector);  
    this.jLabelActionMessage.setText("你将"+countryName+"从列表中删除");  
}else{  
    this.jLabelActionMessage.setText("请先选择一个选项");  
}
```

(13) 将下列代码添加到 Demo 类的 public static void main(String[] args)方法中，并删除原来方法体内的代码。

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoJListFrame().setVisible(true);  
    }  
});
```

至此，已经成功地完成了 JList 程序的开发。

5.1.4 程序功能演示

前面开发了 JList 的程序，本节将对该程序的功能进行演示，按照如下步骤进行。

(1) 编译并运行主项目，窗口如图 5-4 所示。

(2) 在 jTextFieldCountryName 中输入“中国”后，单击“添加”按钮，此时程序如图 5-5 所示。

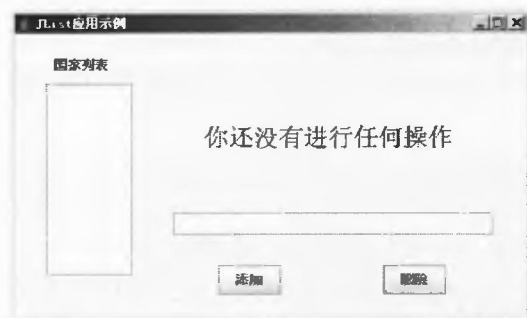


图 5-4 程序运行效果图

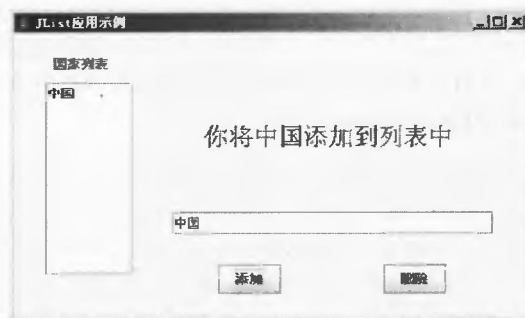


图 5-5 添加“中国”后程序运行效果图

(3) 依次向列表中添加“美国”、“日本”、“英国”、“德国”、“法国”、“意大利”和“韩国”。然后在 `(jTextFieldCountryName)` 中输入一个空字符串，单击“添加”按钮，此时程序运行效果如图 5-6 所示。

(4) 选中列表中的“法国”，程序运行效果如图 5-7 所示。

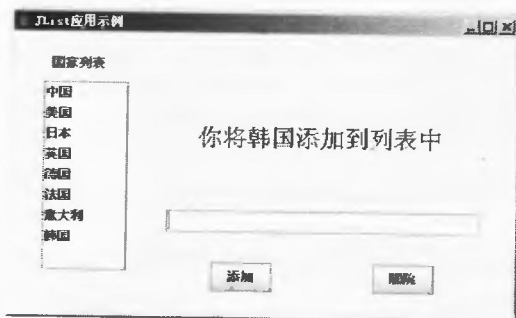


图 5-6 添加空内容时程序运行效果图

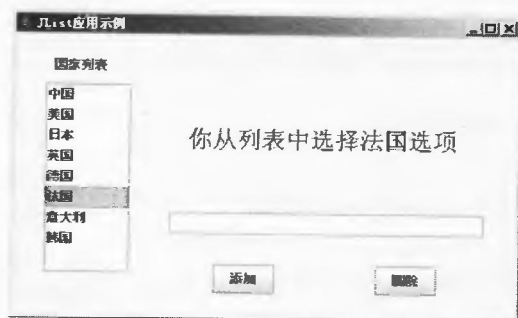


图 5-7 选择“法国”时程序运行效果图

(5) 在图 5-7 窗口中单击“删除”按钮，程序运行效果如图 5-8 所示。

(6) 在图 5-8 窗口中继续单击“删除”按钮，程序运行效果如图 5-9 所示。

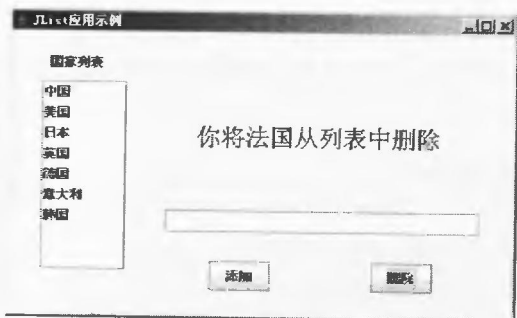


图 5-8 删除“法国”后程序运行效果图

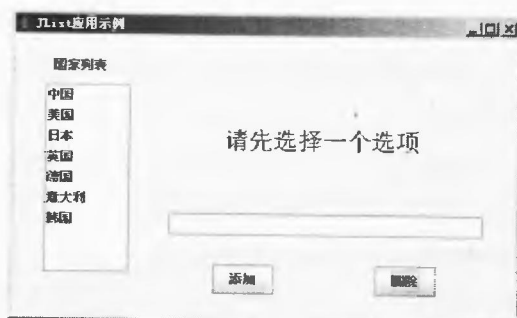


图 5-9 无选中内容时单击“删除”按钮的效果图

5.2 Swing 组合框

`JComboBox` 也是一种可以供用户从中选择的控件，其包括一个按钮和一个下拉列表。本节将对 `JComboBox` 展开讨论，主要包括如下内容：

- `JComboBox` 类简介；
- NetBeans 中 `JComboBox` 的使用。

5.2.1 JComboBox 类简介

`JComboBox` 是一个组合框，其包括一个下拉列表和一个按钮。默认情况下 `JComboBox` 控件提供了一个文本编辑字段，旁边是一个包含向下箭头的小按钮。按下该小按钮之后，会弹出选项列表，用户可以选择其中的一个选项。

用户选择选项以后，该选项的内容会被复制到编辑区域中，弹出式列表将消失。如果以前做出过选择，以前的值会被新值所代替。`JComboBox` 的使用非常简单，以下代码创建了一

个初始内容为空的 JComboBox:

```
JComboBox myJComboBox=new JComboBox();
```

JComboBox 有 4 个构造器, 如表 5-4 所示。

表 5-4 JComboBox 构造器及其说明

构造器	说 明
JComboBox()	创建具有默认数据模型的 JComboBox, 默认的数据模型为空对象列表, 使用 addItem 添加项, 默认情况下, 选择数据模型中的第一项
JComboBox(Object[] items)	创建包含指定数组中元素的 JComboBox。默认情况下, 选择数组中的第一项
JComboBox(Vector items)	创建包含指定 Vector 中的元素的 JComboBox。默认情况下, 选择数组中的第一项
JComboBox(ComboBoxModel model)	创建一个 JComboBox, 其项取自现有的 ComboBoxModel。由于提供了 ComboBoxModel, 使用此构造方法创建的组合框不创建默认组合框模型, 这可能对插入、移除和添加方法的行为都有影响

从表 5-4 中可以看出, JComboBox 可以使用对象数组、向量等作为其对应的 MVC 中的模型。这样, 在更新 JComboBox 中数据的时候, 只需对相应的模型进行操作即可。

JList 提供了很多方法进行相应的操作, 表 5-5 列出了其中常用的方法并给出了简要的说明。

表 5-5 JComboBox 常用的方法及说明

方 法	说 明
setModel(ComboBoxModel aModel)	设置 JComboBox 用于获取项列表的数据模型
getModel()	获取 JComboBox 当前使用的数据模型
setEditable(boolean aFlag)	确定 JComboBox 字段是否可编辑, 可编辑的 JComboBox 允许用户在字段中键入内容或者从列表中选择项来初始化字段, 执行此操作后, 该字段就可编辑了(编辑仅影响字段, 列表项保持原样), 不可编辑的 JComboBox 在字段中显示所选项, 但不能修改该选择
setMaximumRowCount(int count)	设置 JComboBox 显示的最大行数, 如果模型中的对象数大于 count, 则组合框使用滚动条
getMaximumRowCount()	获取组合框不使用滚动条可以显示的最大项数
setSelectedItem(Object anObject)	将所选项设置为参数中的对象。如果 anObject 在列表中, 则显示区域将显示所选的 anObject, 如果 anObject 不在列表中, 且组合框不可编辑, 则不会更改显示区域中的当前选择, 对于可编辑的组合框, 选择将更改为 anObject
getSelectedItem()	获取当前所选项
setSelectedIndex(int anIndex)	选择索引 anIndex 处的项
getSelectedIndex()	获取列表中与给定项匹配的项, 如果 JComboBox 允许选择不在列表中的项, 则结果未必是明确的, 如果不存在所选项或者用户指定的项不在列表中, 则返回-1
addItem(Object anObject)	为项列表添加项。仅当 JComboBox 使用可变数据模型时此方法才有效

续表

方 法	说 明
<code>insertItemAt(Object anObject,int index)</code>	在项列表中的给定索引处插入项, 仅当 JComboBox 使用可变数据模型时此方法才有效
<code>removeItem(Object anObject)</code>	从列表中移除项, 仅当 JComboBox 使用可变数据模型时此方法才有效
<code>removeItemAt(int anIndex)</code>	除 <code>anIndex</code> 处的项, 仅当 JComboBox 使用可变数据模型时此方法才有效
<code>removeAllItems()</code>	项列表中移除所有项
<code>showPopup()</code>	促使组合框显示弹出窗口
<code>hidePopup()</code>	促使组合框关闭弹出窗口
<code>setPopupVisible(boolean v)</code>	设置弹出窗口的可见性
<code>isPopupVisible()</code>	确定弹出窗口的可见性
<code>getItemCount()</code>	获取列表中的项数
<code>getItemAt(int index)</code>	获取指定索引处的列表项, 如果 <code>index</code> 超出范围 (小于零或者大于或等于列表大小), 则其返回 <code>null</code>

另外, JComboBox 还继承了其父类的 `javax.swing.JComponent` 的大量方法, 这里不再赘述, 有兴趣的读者可以查阅相关的 API。

当列表中的当前选择发生变化时, 会触发 `ItemEvent` 事件。当用户在按钮的下拉列表框中做出选择时, 会触发 `ActionEvent` 事件。可以使用标准的事件处理方法来处理这些事件。

如果开发中需要通过代码增删下拉列表中的选项, 也可以使用 MVC 中的模型来进行管理, 非常方便, 具体方法如下:

提示

```
v.add("Item 12"); //其中 v 为向量模型对象
myJComboBox.setModel(new DefaultComboBoxModel(v));
```

5.2.2 NetBeans 中 JComboBox 的使用

在 NetBeans 中可以很方便地创建并使用 JComboBox, 在本小节中, 将通过一个例子来介绍在 NetBeans 使用 JComboBox 的方法。打开上一章中创建的项目 `container`, 按照如下步骤完成该例。

(1) 向其中添加一个使用 JFrame Form 模版创建的 `DemoJComboBoxFrame` 类。

(2) 添加完成以后, 打开 GUI 设计器, 在 Palette 窗口中选择 “JComboBox” 选项, 之后在 GUI 设计器中的适当位置单击鼠标左键, 即可向 `DemoJComboBoxFrame` 类添加一个 JComboBox。

(3) 用鼠标右键单击该 JComboBox, 选择右键菜单中的 Change Variable Name 选项, 在弹出的对话框中将该 JComboBox 的名称修改为 `JComboBoxLover`。

(4) 选中 `JComboBoxLover`, 在属性对话框中修改其 `model` 属性, 删除其原来的所有选项后, 添加一个空选项, 接着再依次添加 `football`、`basketball`、`vollyball`、`swaming danceing`、`singing` 等。

(5) 选中 `JComboBoxLover`, 单击鼠标右键, 在弹出的菜单中依次选择 “Events” /

“ItemEvent” / “itemStateChanged” 选项。为 JComboBoxLover 添加 ItemEvent 事件的处理方法。此时，源代码编辑器中出现如图 5-10 所示内容。

```
private void jComboBoxLoverItemStateChanged(java.awt.event.ItemEvent evt) {  
    // TODO add your handling code here:  
}
```

图 5-10 JComboBox 的 LoveItemEvent 事件处理方法

(6) 向 DemoJComboBoxFrame 类添加一个 JLabel，将其名称修改为 jLabelMessage，并将其 text 属性修改为“请从列表选择一个选项”。

(7) 此时，GUI 设计器中内容如图 5-11 所示。

(8) 将下列代码添加到 jComboBoxLoverItemStateChanged 方法的方法中：

```
String favor=(String)this.jComboBoxLover.getSelectedItem();  
this.jLabelMessage.setText("你喜欢"+favor);
```

(9) 将下列代码添加到 Demo 类的 public static void main(String[] args)方法中，并删除原来方法体内的代码：

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoJComboBoxFrame ().setVisible(true);  
    }  
});
```

(10) 编译并运行主项目，窗口内容如图 5-12 所示。

(11) 从 JComboBoxLover 中选择“football”选项，此时窗口内容如图 5-13 所示。

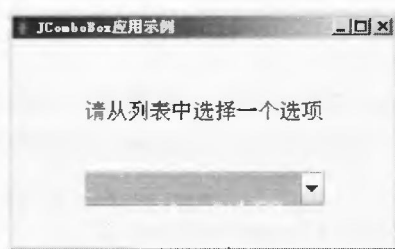


图 5-12 程序运行效果图

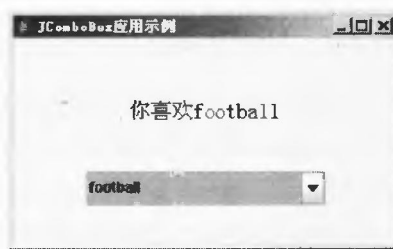


图 5-13 选择“football”效果图

从 JComboBoxLover 中选择不同的选项，在 jLabelMessage 上会显示不同的内容，有兴趣的读者可以通过自己实验来观察现象，这里就不一一列出了。

5.3 Swing 分割窗口

JSplitPane 也是 Swing 中的常用控件之一，其将单个窗格分割成两部分。本节将对 JSplitPane 展开讨论，主要包括如下内容：

- JSplitPane 类简介;
- NetBeans 中 JSplitPane 的使用。

5.3.1 JSplitPane 类简介

JSplitPane 可以将单个窗格拆分为两部分。开发人员可以决定是按水平方向还是按垂直方向拆分窗格,用户在程序运行期间还可以调整窗格的拆分比例。通过嵌套使用 JSplitPane,可以将窗口中内容分割成 3 个或更多部分。

创建一个 JSplitPane, 代码设置如下:

```
JSplitPane myJSplitPane=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
jSplitPaneLeft, jSplitPaneRight);
```

提示 jSplitPaneLeft 和 jSplitPaneRight 分别指两个控件的对象句柄。

对于 JSplitPane 中的每一部分,只能向其中添加一个控件。如果需要添加的控件多于一,可以先向其中的每一部分添加一个 JPanel,然后再向 JPanel 中添加控件。JSplitPane 有 5 个构造器,如表 5-6 所示。

表 5-6 JSplitPane 构造器及说明

构造器	说 明
JSplitPane()	创建一个配置为将其子组件水平排列、无连续布局、为组件使用两个按钮的新 JSplitPane
JSplitPane(int newOrientation)	创建一个配置为指定方向且无连续布局的新 JSplitPane
JSplitPane(int newOrientation, boolean newContinuousLayout)	创建一个具有指定方向和重绘方式的新 JSplitPane
JSplitPane(int newOrientation, Component newLeftComponent, Component newRightComponent)	创建一个具有指定方向和不连续重绘的指定组件的新 JSplitPane
JSplitPane(int newOrientation, boolean newContinuousLayout, Component newLeftComponent, Component newRightComponent)	创建一个具有指定方向、重绘方式和指定组件的新 JSplitPane

在表 5-6 中所使用到的 newOrientation 的可选值为 JSplitPane.HORIZONTAL_SPLIT 或 JSplitPane.VERTICAL_SPLIT。

另外 JSplitPane 还提供了一些与分割窗体有关的方法,如表 5-7 所示。

表 5-7 JSplitPane 的常用方法及说明

方 法	说 明
setDividerSize(int newSize)	设置分隔条的大小
getDividerSize()	获得分隔条的大小
setLeftComponent(Component comp)	将组件设置到分隔条的左边(或上面)
Component getLeftComponent()	获得分隔条左边(或者上面)的组件
setTopComponent(Component comp)	将组件设置到分隔条的上面或者左边

续表

方 法	说 明
Component getTopComponent()	获得分隔条上面或者左边的组件
setRightComponent(Component comp)	将组件设置到分隔条的右边（或者下面）
Component getRightComponent()	获得分隔条右边（或者下面）的组件
setBottomComponent(Component comp)	将组件设置到分隔条的下面或者右边
Component getBottomComponent()	获得分隔条下面或者右边的组件
setLastDividerLocation(int newLastLocation)	将分隔条所处的最后位置设置为 newLastLocation
getLastDividerLocation()	获得分隔条所处的最后位置
getOrientation()	获得方向
setDividerLocation(double proportionalLocation)	按照百分比设置分隔条的位置
setDividerLocation(int location)	按照百分比设置分隔条的位置
getDividerLocation()	获得最后传递给 setDividerLocation 的值，此方法返回的值可能与实际的分隔条位置不同
getMinimumDividerLocation()	获得外观实现中分隔条的最小位置
getMaximumDividerLocation()	获得外观实现中分隔条的最大位置



提示 如果将 DividerSize 的值设置为 0，则使用 JSplitPane 分割出的两部分不能在运行时通过鼠标拖曳改变大小。

5.3.2 NetBeans 中 JSplitPane 的使用

在 NetBeans 中使用 JSplitPane 是一个非常简单的过程，本节通过一个例子展示在 NetBeans 中使用 JSplitPane 的方法。打开上一章中创建的项目 container，按照如下步骤完成该例。

- (1) 向其中添加一个使用 JFrame Form 模版创建的 DemoJSplitPaneFrame 类。
- (2) 添加完成以后，打开 GUI 设计器，在 Palette 窗口中选择“Jpanel”选项，之后在 GUI 设计器中的适当位置单击鼠标，即向 DemoJSplitPaneFrame 中添加一个 JPanel。
- (3) 选中 JPanel，并将其名称修改为“jPanelGlobal”。
- (4) 接着在 Palette 窗口中选择“JSplitPane”选项，并在 jPanelGlobal 中的适当位置单击鼠标，即向 DemoJSplitPaneFrame 类中的 JPanel 添加一个 JSplitPane。将该 JSplitPane 的名称修改为“JSplitPaneGlobal”。
- (5) 在属性对话框中修改 JSplitPaneGlobal 的属性，将 dividerLocation 的值设置为 200，dividerSize 的值设置为 3。此时 GUI 设计器内容如图 5-14 所示。



图 5-14 分割窗口

- (6) 在 Palette 窗口中选择“Jlabel”选项，之后在 JSplitPaneGlobal 的“左键”所标识的

位置单击鼠标，即向 `DemoJSplitPaneFrame` 类添加一个 `JLabel`。将该 `JLabel` 的名称修改为“`jLabelLeft`”，并将其 `text` 属性值修改为“这是 `jSplitPaneGlobald` 的左部”。

(7) 按上述步骤再向 `jSplitPaneGlobald` 的“右键”标识部分添加一个 `JLabel`，将该 `JLabel` 的名称修改为“`jLabelRight`”，并将其 `text` 属性值修改为“这是 `jSplitPaneGlobald` 的右部”。

(8) 接着向 `DemoJSplitPaneFrame` 类添加一个名称为“`jButtonReset`”的“`Jbutton`”。将其 `text` 属性值修改为“还原”，并为其添加 `ActionEvent` 事件的处理方法 `jButtonResetActionPerformed`。并将下列代码添加到 `jButtonResetActionPerformed` 方法的方法中：

```
this.jSplitPaneGlobal.setDividerLocation(130);
```

(9) 将下列代码添加到 `Demo` 类的 `public static void main(String[] args)` 方法中，并删除原来方法中的代码：

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoJSplitPaneFrame().setVisible(true);  
    }  
});
```

(10) 编译并运行主项目，运行界面如图 5-15 所示。

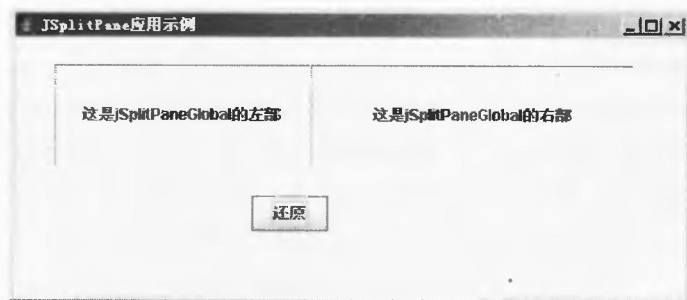


图 5-15 程序运行效果图

(11) 在如图 5-15 所示的窗口中，拖动分割条，可以调整左右两部分的大小。调整后如果单击“还原”按钮，则窗口又会还原为如图 5-15 所示的情况。

5.4 实战：使用 NetBeans 构建 Swing 的综合例程

本章前面的小节中介绍了 `Swing` 中各种常用控件的使用方法。本节将开发一个综合使用各种 `Swing` 控件的项目。

5.4.1 项目概述

在本节中将使用 `NetBeans` 开发一个动作列表程序，程序界面如图 5-16 所示。

在如图 5-16 所示的界面中，可以在列表中选择选项，也可以选择各个按钮，还可以在地址栏中输入内容并按下回车键。这些都能够触发相应的动作事件，并且程序会在“这里将显示动作”所示的区域以文本的形式对相应的动作进行说明。

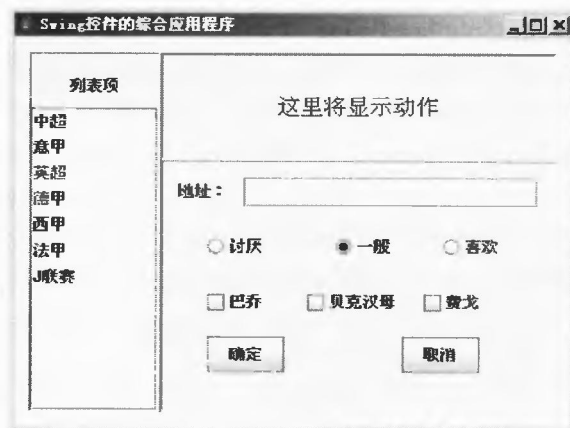


图 5-16 程序运行界面

5.4.2 界面设计

启动 NetBeans，按照如下步骤完成界面设计。

- (1) 创建一个名称为“actionlist”的项目，并将其主类类名设置为 `org.netbeans.swing.actionlist.ActionList`。
- (2) 向 actionlist 添加一个通过 JFrame Form 模版创建的 ActionListFrame 类。
- (3) 向 ActionListFrame 中添加一个 JSplitPane，将其名称修改为“jSplitPaneGlobal”，并将 `dividerLocation` 属性的值设置为 100，`dividerSize` 属性的值设置为 3。
- (4) 向 jSplitPaneGlobal 的“右键”所标识的部分添加 JSplitPane，将其名称修改为 jSplitPaneRight。设置该 JSplitPane 的 `orientation` 属性为“VETICAL_SPLIT”，并将 `dividerLocation` 属性和 `dividerSize` 属性分别设置为 80 和 3。此时界面如图 5-17 所示。

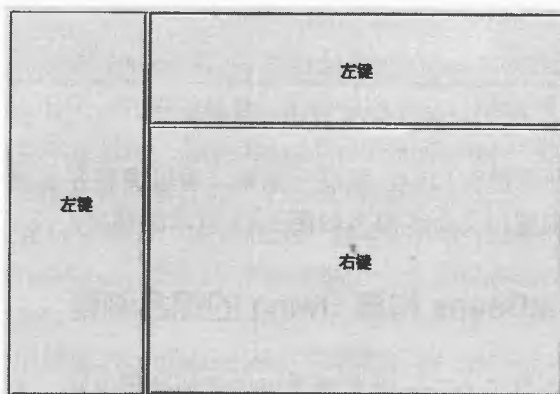


图 5-17 分割窗格界面

- (5) 分别向 jSplitPaneGlobal 的“左键”标识部分，jSplitPaneRight 的“左键”和“右键”标识部分添加一个 JPanel，并依次将其名称修改为“jPanelLeft”、“jPanelRightTop”和“jPanelRightBottom”。
- (6) 向 jPanelLeft 添加一个 JLabel，将其名称修改为 jLabelList，并将其 `test` 属性的值修改为“列表项”。
- (7) 接着向 jPanelLeft 添加一个 JList，将其名称修改为“jListSport”。设置 jListSport 的

初始选项为“中超”、“意甲”、“英超”、“德甲”、“西甲”、“法甲”和“J联赛”。

(8) 向 jPanelRightTop 添加一个 JLabel, 将其名称修改为“jLabelMessage”, 并设置其 test 属性的值为“这里将显示动作”。

(9) 向 jPanelRightBottom 添加一个 JLabel, 将其名称修改为“jLabelAddress”, 并设置其 test 属性的值为“地址:”。然后向 jPanelRightBottom 添加一个 JTextField, 将其名称修改为 jTextFieldAddress, 并设置其 test 属性的值为空。

(10) 向 jPanelRightBottom 添加一个 ButtonGroup, 将其名称修改为 buttonGroupOne。

(11) 依次向 jPanelRightBottom 添加 1 个 JLabel, 1 个 JTextField, 2 个 JButton, 3 个 JRadioButton 和 3 个 JCheckBox, 并按如表 5-8 所示修改其属性。

表 5-8 控件与属性值对照表

控件类型	控件名称	text 属性值
JLabel	jLabelAddress	地址:
JTextField	jTextFieldAddress	空
JRadioButton	jRadioButtonFavor	喜欢
JRadioButton	jRadioButtonCommon	一般
JRadioButton	jRadioButtonDislike	讨厌
JCheckBox	jCheckBoxBaggio	巴乔
JCheckBox	jCheckBoxBeakHam	贝克汉母
JCheckBox	jCheckBoxFigo	费戈
JButton	jButtonEnter	确定
JButton	jButtonCancel	取消

(12) 依次将 jRadioButtonFavor、jRadioButtonCommon、jRadioButtonDislike 的 buttonGroup 属性的值设置为 buttonGroupOne, 并将 jRadioButtonCommon 的 selected 属性设置为选中状态。

该程序的界面部分的设计到此已经成功地完成了, 此时界面如图 5-18 所示。

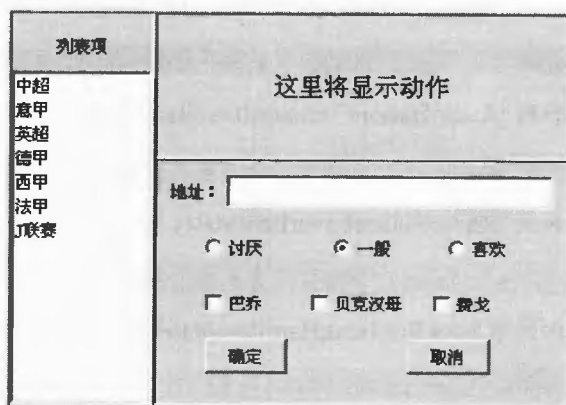


图 5-18 程序界面

5.4.3 功能代码的开发

完成界面设计以后, 进入功能代码的开发阶段。按照如下步骤完成代码开发。

(1) 按照如表 5-9 所示, 为控件添加事件处理方法。

表 5-9 控件事件及其事件处理方法

控件名称	事件名称	方法名称
jListSport	ListSelection	valueChanged
jTextFieldAddress	ActionEvent	actionPerformed
JRadioButton	jRadioButtonFavor	itemStateChanged
JRadioButton	jRadioButtonCommon	itemStateChanged
JRadioButton	jRadioButtonDislike	itemStateChanged
JCheckBox	jCheckBoxBaggio	itemStateChanged
JCheckBox	jCheckBoxBeakHam	itemStateChanged
JCheckBox	jCheckBoxFigo	itemStateChanged
JButton	jButtonEnter	actionPerformed
JButton	jButtonCancel	actionPerformed

(2) 按照下面的步骤, 向相应的事件处理方法内添加代码。

- 将下列代码添加到 jListSportValueChanged 方法的方法中:

```
String sport=(String)this.jListSport.getSelectedValue();
this.jLabelMessage.setText("你选择了"+ sport);
```

- 将下列代码添加到 jTextFieldAddressActionPerformed 方法的方法中:

```
String address=this.jTextFieldAddress.getText();
if(address!=null&&!address.equals("")) {
    this.jLabelMessage.setText("你输入了"+address);
}else{
    this.jLabelMessage.setText("你没有输入任何内容");
}
```

- 将下列代码添加到 jRadioButtonDislikeItemStateChanged 方法中:

```
this.jLabelMessage.setText("你选择了讨厌单选按钮");
```

- 将下列代码添加到 jRadioButtonCommonItemStateChange 方法中:

```
this.jLabelMessage.setText("你选择了普通单选按钮");
```

- 将下列代码添加到 jRadioButtonFavorItemStateChanged 方法中:

```
this.jLabelMessage.setText("你选择了喜欢单选按钮");
```

- 将下列代码添加到 jCheckBoxBeakHamItemStateChanged 方法中:

```
if(this.jCheckBoxBeakHam.isSelected()){
    this.jLabelMessage.setText("你开始喜欢贝克汉母了");
}else{
    this.jLabelMessage.setText("你不喜欢贝克汉母了");
}
```

- 将下列代码添加到 jCheckBoxFigoItemStateChanged 方法中:

```
if(this.jCheckBoxFigo.isSelected()){
this.jLabelMessage.setText("你开始喜欢费戈了");
}else{
this.jLabelMessage.setText("你不喜欢费戈了");
}
```

- 将下列代码添加到 `jCheckBoxBaggioItemStateChanged` 方法中:

```
if(this.jCheckBoxBaggio.isSelected()){
this.jLabelMessage.setText("你开始喜欢巴乔了");
}else{
this.jLabelMessage.setText("你不喜欢巴乔了");
}
```

- 将下列代码添加到 `jButtonEnterActionPerformed` 方法中:

```
this.jLabelMessage.setText("你按下了确定按钮");
```

- 将下列代码添加到 `jButtonCancelActionPerformed` 方法中:

```
this.jLabelMessage.setText("你按下了取消按钮");
```

- (3) 将下列代码添加到 `ActionList` 类的 `public static void main(String args[])` 方法中:

```
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new ActionListFrame().setVisible(true);
    }
});
```

- (4) 编译并运行主程序,即可得到本节开始给出的如图 5-16 所示的界面。

- (5) 选中“巴乔”复选框后,界面如图 5-19 所示。

- (6) 选中列表中的“中超”选项,界面如图 5-20 所示。

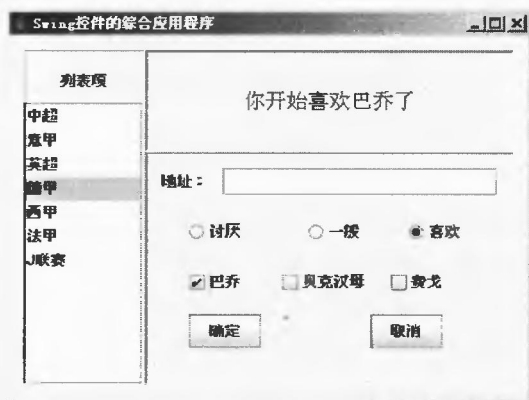


图 5-19 选中“巴乔”复选框

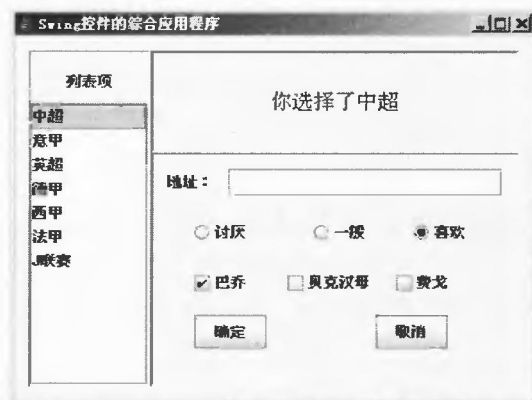


图 5-20 选中“中超”选项

通过选择不同的选项,会得到很多不同的结果,此处就不一一给出了,有兴趣的读者可以自己实验,以观察不同的运行效果。

5.5 小结

在上一章学习的基础上，本章结合具体的实例，为读者讲述了另外几种 Swing 常用控件，包括列表、组合框和分割窗口。并开发了一个综合使用各种 Swing 控件的项目，以提高读者综合使用各种控件的能力。

第 6 章

Swing 容器类

上一章为读者介绍了 Swing 中一些常用控件, 及其在 NetBeans 中的使用, 从中可以看到, 这些控件并不能孤立存在, 而是必须要添加到像 JFrame 这样的“面板”中。在 Java 中, 用来存放控件的“面板”被称为容器。在本章将介绍 Java 中提供一些容器类, 并详细地讲述了在 NetBeans 中使用这些容器类的方法。主要包括如下内容:

- JFrame;
- JApplet;
- JWindow。

6.1 Swing 框架——JFrame

JFrame 是图形用户界面编程中常用到的 Swing 容器类。本节将对 JFrame 展开讨论, 主要包括如下内容:

- JFrame 类简介;
- NetBeans 中 JFrame 的使用。

6.1.1 JFrame 类简介

JFrame 类扩展自 java.awt.Frame。与 java.awt.Frame 类似, JFrame 提供了一个包含标题、边框和平台专用装饰 (如最大化按钮、最小化按钮、关闭按钮) 的顶层窗口。

在 JDK 1.5 以前的版本中向 JFrame 中添加控件与 Frame 稍有不同, 以下代码显示了如何创建一个 JFrame 并向其中添加一个 JLabel。

```
JLabel myJLabel=new JLabel("使用 JFrame");  
JFrame myJFrame=new JFrame();  
myJFrame.getContentPane().add(myJLabel);
```

如果使用 JDK 1.5, 则可以直接调用 add 方法向 JFrame 中添加控件。下面代码显示了 JDK

1.5 中如何创建一个 JFrame，并向其中添加一个 JLabel。

```
JLabel myJLabel=new JLabel("使用 JFrame");
JFrame myJFrame=new JFrame();
myJFrame.add(myJLabel);
```

提示 即使在 JDK 1.5 中，添加到 JFrame 的控件实质上还是被添加到 JFrame 的 ContentPane 中，只不过该过程由系统自动完成，不需要开发人员完成。

JFrame 类有 4 个构造器，如表 6-1 所示。

表 6-1 JFrame 的构造器及其说明

构造器	说 明
JFrame()	构造一个初始时不可见的 JFrame
JFrame(String title)	创建一个新的、初始不可见的、具有指定标题的 JFrame
JFrame(GraphicsConfiguration gc)	以屏幕设备的指定 GraphicsConfiguration 和空白标题创建一个 JFrame
JFrame(String title, GraphicsConfiguration gc)	创建一个具有指定标题和指定屏幕设备的 JFrame

新创建的 JFrame，其默认情况下是不可见的，这使得开发人员有机会在显示 JFrame 之前进行一些操作。可以使用 setVisible 方法设置其可见性。

JFrame 类可以使用很多方法进行相应的操作，表 6-2 列出了 JFrame 中常用的方法并给出了简要的说明。

表 6-2 JFrame 的常用方法及说明

方 法	说 明
processWindowEvent(WindowEvent e)	处理此组件上发生的窗口事件。隐藏窗口或释放窗口，根据 defaultCloseOperation 属性的设置来指定
setDefaultCloseOperation(int operation)	设置用户在此窗体上发起“close”时，默认执行的操作
getDefaultCloseOperation()	获得用户在此窗体上发起“close”时，执行的操作
update(Graphics g)	只是调用 paint(g)。重写此方法，以防止不必要的调用清除背景
setJMenuBar(JMenuBar menubar)	设置此窗体的菜单栏
getJMenuBar()	获得此窗体上设置的菜单栏
isRootPaneCheckingEnabled()	获得是否将对 add 和 setLayout 的调用转发到 contentPane
setRootPaneCheckingEnabled(boolean enabled)	设置是否将对 add 和 setLayout 的调用转发到 contentPane
setLayout(LayoutManager manager)	设置 JFrame 的布局管理器
getContentPane()	获得此窗体的 contentPan 对象
setTitle(String title)	设置此 JFrame 标题栏中的文字
dispose()	关闭 JFrame 并回收用于创建窗口的任何资源
setResizable(boolean b)	使用一个 boolean 参数来决定 JFrame 的大小是否允许用户改变
setIconImage(Image image)	设置标题栏的图像
setVisible(boolean b)	设置 JFrame 的可见性

续表

方 法	说 明
<code>setAlwaysOnTop(boolean alwaysOnTop)</code>	更改始终位于顶层的窗口状态。始终位于顶层的窗口是位于所有其他窗口之上的窗口。如果存在多个始终位于顶层的窗口，则不指定它们相互的叠放顺序，而且顺序与平台有关
<code>isAlwaysOnTop()</code>	获得此窗口是否为 <code>always-on-top</code> 窗口

使用 `getDefaultCloseOperation()` 与 `setDefaultCloseOperation(int operation)` 获取和设置 `defaultCloseOperation` 属性，其值必须为下列几种之一。

- `DO_NOTHING_ON_CLOSE` (在 `WindowConstants` 中定义): 不执行任何操作; 要求程序在已注册的 `WindowListener` 对象的 `windowClosing` 方法中处理该操作。
- `HIDE_ON_CLOSE` (在 `WindowConstants` 中定义): 调用任意已注册的 `WindowListener` 对象后自动隐藏该窗体。
- `DISPOSE_ON_CLOSE` (在 `WindowConstants` 中定义): 调用任意已注册 `WindowListener` 的对象后自动隐藏并释放该窗体。
- `EXIT_ON_CLOSE` (在 `JFrame` 中定义): 使用 `System.exit` 方法退出应用程序。仅在应用程序中使用。

提示 上述两个方法是在 JDK 1.3 中引入的，如果当前正使用一个早期版本，则这两个方法不可用。

另外，`JFrame` 还能够使用很多从父类继承的方法，如设置位置、大小等。需要用到的用户可以查阅相关的 API，这里不再赘述。

提示 如果不明确地设置 `JFrame` 的大小，`JFrame` 的初始大小为 0 像素×0 像素。应该把 `JFrame` 的大小设置为大多数显示设备所接受的大小。

6.1.2 NetBeans 中 JFrame 的使用

在前面的章节中，实际上已经多次在 NetBeans 中使用了 `JFrame`。本节将通过一个例子，来深入地讨论 NetBeans 中 `JFrame` 的使用。启动 NetBeans，按照如下步骤完成该例。

- (1) 新建一个名称为 `component` 的项目，并将其主类名称修改为 `org.netbeans.swing.component.Demo`。
- (2) 向项目中添加一个使用 `JFrame Form` 模版创建的 `JFrame`，并将其名称修改为“`DemoJFrame`”。
- (3) 在 GUI 设计器中选中的 `DemoJFrame`，根据如表 6-3 所示的内容在属性对话框中设置相应的属性。

表 6-3 属性及其值对照表

属 性 名 称	属 性 值	作 用
<code>title</code>	<code>JFrame</code> 应用示例	设置该 <code>JFrame</code> 标题栏中文字
<code>alwaysOnTop</code>	<code>true</code>	设置该窗口是否位于最上端
<code>resizable</code>	<code>true</code>	是否允许改变该 <code>JFrame</code> 大小

(4) 向 DemoJFrame 添加一个 JLabel, 将其名称修改为 “jLabelMessage”, 并将其 text 属性的值修改为 “这里将显示 JFrame 的属性”。

(5) 向 DemoJFrame 添加一个 JList, 将其名称修改为 “jListSize”, 并设置其初始选项为 300×200、400×300、600×450、800×600。

(6) 接着向 DemoJFrame 添加两个 JCheckBox, 分别将其名称修改为 “jCheckBoxSize”、“jCheckBoxOnTop”, 并将其 text 属性的值设置为 “可以更改大小”、“总在最前面”。最后把这两个 JCheckBox 的 selected 属性设置为选中状态。此时, 界面如图 6-1 所示。

(7) 为 jListSize 添加 ListSelectionEvent 事件的处理方法, 并分别为 jCheckBoxSize 和 jCheckBoxOnTop 添加 ItemEvent 事件的处理方法。

(8) jCheckBoxSizeItemStateChanged 方法用来处理 jCheckBoxSize 的 ItemEvent 事件, 将下列代码添加到该方法中:

```
if(this.jCheckBoxSize.isSelected()){
    this.setResizable(true);
    this.jListSize.setEnabled(true);
    this.jLabelMessage.setText("当前窗口可以改变大小");
}else{
    this.setResizable(false);
    this.jListSize.setEnabled(false);
    this.jLabelMessage.setText("当前窗口不可以改变大小");
}
```

- 以上代码的作用是设置该 JFrame 是否可以改变大小。
- 在 jLabelMessage 上显示相应的状态信息。

(9) jCheckBoxOnTopItemStateChanged 方法用来处理 jCheckBoxSize 的 ItemEvent 事件, 将下列代码添加到该方法中:

```
if(this.jCheckBoxOnTop.isSelected()){
    this.setAlwaysOnTop(true);
    this.jLabelMessage.setText("当前窗口总位于最前面");
}else{
    this.setAlwaysOnTop(false);
    this.jLabelMessage.setText("当前窗口不是总位于最前面");
}
```

- 以上代码的作用是设置该 JFrame 是否总位于屏幕的最上面。
- 在 jLabelMessage 上显示相应的状态信息。

(10) jListSizeValueChanged 方法用来处理 jListSize 的 ListSelectionEvent 事件, 将下列代码添加到该方法中:

```
int frameSize=this.jListSize.getSelectedIndex();
switch(frameSize){
```

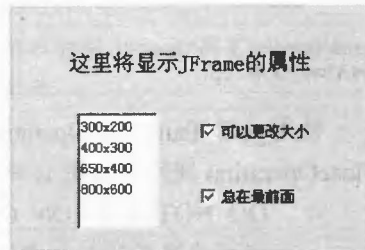


图 6-1 JFrame 示例界面

```
case 0:
    this.setSize(300,200);
    this.jLabelMessage.setText("当前窗口大小为 300x200");
    break;
case 1:
    this.setSize(400,300);
    this.jLabelMessage.setText("当前窗口大小为 400x300");
    break;
case 2:
    this.setSize(650,400);
    this.jLabelMessage.setText("当前窗口大小为 650x400");
    break;
case 3:
    this.setSize(800,600);
    this.jLabelMessage.setText("800x600");
    break;
}
}
```

(11) 将下列代码添加到 Demo 类中的 `public static void main(String[] args)` 方法中:

```
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoJFrame().setVisible(true);
    }
});
```

(12) 编译并运行主项目, 此时界面如图 6-2 所示。此时窗口总是位于最前面, 并且可以通过鼠标拖动来改变窗口的大小, 如图 6-3 所示。

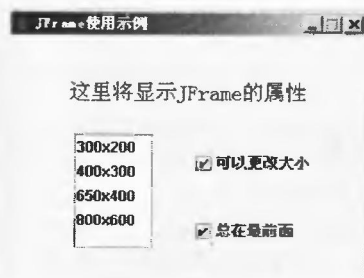


图 6-2 程序运行效果图

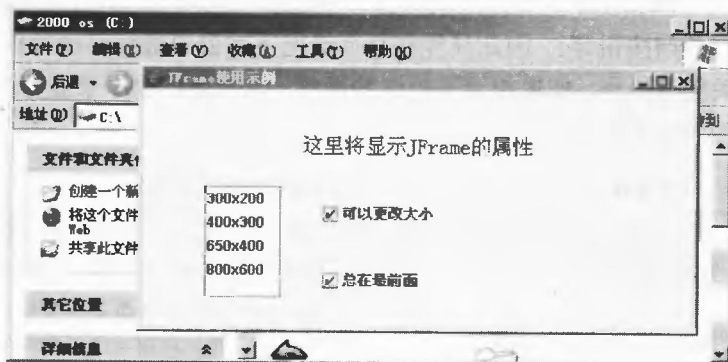


图 6-3 改变大小及总位于最前面

(13) 选择列表框中的选项, 可以改变窗口的大小。选择 400x300 选项后, 界面如图 6-4 所示。

(14) 将“可以更改大小”复选框设置为非选中状态, 此时界面如图 6-5 所示。

在此状态下, 窗体的大小不可以改变。本程序还有其他运行状态, 这里就不一一给出了, 有兴趣的读者可以运行程序, 并查看不同的运行效果。

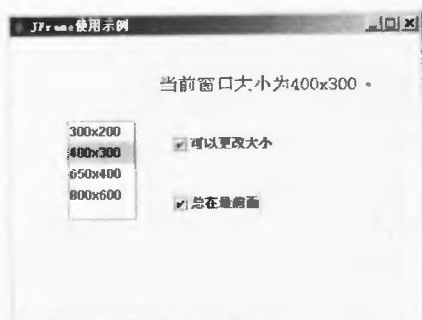


图 6-4 调整窗口大小为 400x300

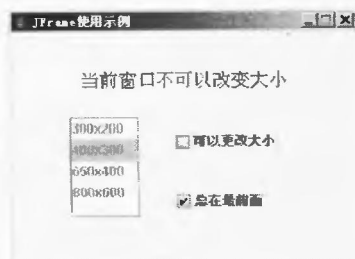


图 6-5 不可以改变大小

6.2 Swing 窗口——JWindow

JWindow 也是 Swing 编程中非常有用的一种容器类。本章将对 JWindow 展开讨论，主要包括如下内容：

- JWindow 类简介；
- 在 NetBeans 中使用 Jwindow。

6.2.1 JWindow 类简介

JWindow 也是一个窗口，但是没有标题栏、窗口管理按钮或者其他与 JFrame 关联的修饰，其可以存在于桌面上的任何位置。JWindow 扩展了 java.awt.Window，基本上没有改动 Window 中定义的内容。以下代码创建了一个简单的 Jwindow：

```
JWindow myJWindow=new JWindow();
```

JWindow 主要是用来在窗口中显示一些内容，并且不希望添加装饰（如最大化、最小化和关闭按钮等）。因此只能在某些情况下对窗口进行移动操作，而不能进行最大化、最小化和关闭等操作。如果想要执行这些操作，只能通过程序来进行。JWindow 类提供了 5 个构造器，如表 6-4 所示。

表 6-4 JWindow 类构造器及其说明

构造器	说 明
JWindow()	创建未指定所有者的窗口，此窗口不可获得焦点
JWindow(Frame owner)	使用指定的所有者框架创建窗口。如果 owner 为 null，则使用共享所有者，而且此窗口不可获得焦点。只有其所有者正显示在屏幕上时此窗口才可获得焦点
JWindow(Window owner)	使用指定的所有者窗口创建窗口。只有在其所有者正显示在屏幕上时此窗口才可获得焦点。如果 owner 为 null，则使用共享所有者，而且此窗口不可获得焦点
JWindow(GraphicsConfiguration gc)	使用屏幕设备的指定 GraphicsConfiguration 创建窗口，此窗口不可获得焦点
JWindow(Window owner, GraphicsConfiguration gc)	使用屏幕设备的指定所有者窗口和 GraphicsConfiguration 创建窗口。如果 owner 为 null，则使用共享所有者，而且此窗口不可获得焦点

作为一个优秀的程序，应该检查用户屏幕的分辨率并且编写代码，把 JWindow 设置为相应的值。因为在一个 PDA 上正常显示的窗口在电脑的显示器上，很可能已经面目全非。

可以使用很多方法对 JWindow 类进行操作，表 6-5 列出了常用的一些方法及其说明。

表 6-5 JWindow 类的常用方法及其说明

方 法	说 明
toFront()	如果此窗口是可见的，则将此窗口置于前端，并可以将其设为焦点窗口。如果此窗口不可见，则不会发生任何操作
toBack()	如果此窗口是可视的，则将此窗口发送到后台，如果它是焦点窗口或活动窗口，则会导致焦点的丢失或激活状态。如果此窗口不可见，则不发生任何操作
getOwner()	获取此窗口的所有者
getOwnedWindows()	获取包含此窗口当前拥有的所有窗口的数组
addWindowListener(WindowListener wl)	添加指定的窗口侦听器，以从此窗口接收窗口事件。如果 wl 为 null，则不抛出异常，而且不执行任何操作
addWindowStateListener(WindowStateListener wsl)	添加指定的窗口状态侦听器，以从此窗口接收窗口事件。如果 wsl 为 null，则不抛出异常，而且不执行任何操作
addWindowFocusListener(WindowFocusListener l)	添加指定的窗口焦点侦听器，以从此窗口接收窗口事件
removeWindowListener(WindowListener l)	移除指定的窗口侦听器，以便不再从此窗口接收窗口事件
removeWindowStateListener(WindowStateListener l)	移除指定的窗口状态侦听器，以便不再从此窗口接收窗口事件
removeWindowFocusListener(WindowFocusListener l)	移除指定的窗口焦点侦听器，以便不再从此窗口接收窗口事件
getWindowListeners()	获取在此窗口注册的所有窗口侦听器的数组
getWindowFocusListeners()	获取在此窗口注册的所有窗口焦点侦听器的数组
getWindowStateListeners()	获取在此窗口注册的所有窗口状态侦听器的数组
getFocusOwner()	如果此窗口为焦点窗口，则获取其具有焦点窗口的子组件；否则返回 null
isActive()	此窗口是否为活动窗口。仅有一个框架或对话框可以处于激活状态
isFocused()	此窗口是否为焦点窗口。如果存在焦点所有者，则焦点窗口就是（或者包含）焦点所有者的窗口。如果不存在焦点所有者，则没有作为焦点的窗口

另外 JWindow 还可以使用父类中提供的很多方法，需要用到这些方法的用户可以查阅相关的 API。

6.2.2 在 NetBeans 中使用 JWindow

因为 JWindow 与 JFrame 的用法存在很大的不同，因此 NetBeans 并没有为 JWindow 提供

可以向其中拖曳控件的 GUI 设计器。NetBeans 允许开发人员自己通过扩展 JWindow 类来开发使用 JWindow 的程序。

JWindow 的一个重要应用就是制作程序的欢迎界面。本节将通过这样的例子来讲述如何在 NetBeans 中使用 JWindow。打开上一节中创建的项目 component，按照如下步骤完成该例。

(1) 在项目窗口中选中 org.netbeans.swing.component 节点，单击鼠标右键，在弹出的菜单中依次选择“New”/“Java Class”选项，弹出如图 6-6 所示的窗口。

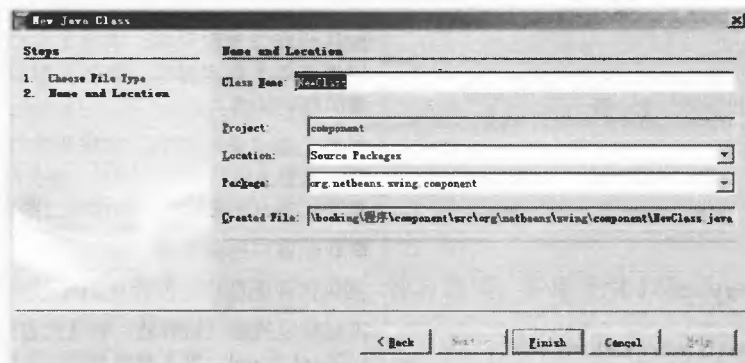


图 6-6 添加类窗口

(2) 将 Class Name 的值设置为 DemoJWindow，该值即为添加类的类名，单击“Finish”按钮完成添加。

(3) 向 DemoJWindow.java 文件中添加类。在项目窗口中选中 DemoJWindow 节点，单击鼠标右键，在弹出的菜单中依次选择“Add”/“Class”选项，打开如图 6-7 所示的窗口。

(4) 在如图 6-7 所示的窗口中可以设置要添加类的名称、父类类型、访问限制修饰符。在此例中，将 Name 的值设置为 WelcomeWindow，将 Super Class 的值设置为 javax.swing.JWindow，并将 final 设置为非选中状态，单击“OK”按钮完成添加。

(5) 此时在源代码编辑器中会出现如图 6-8 所示的内容。

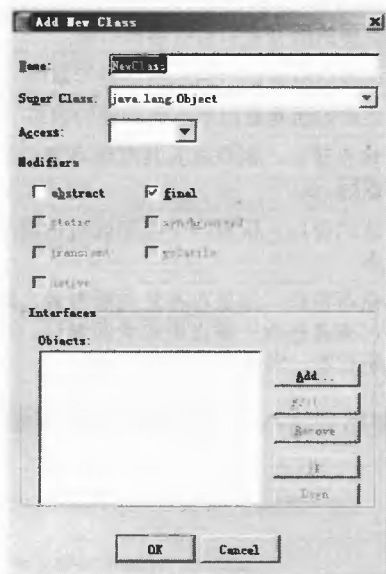


图 6-7 向文件添加类窗口

```
class WelcomeWindow extends javax.swing.JWindow {  
}
```

图 6-8 源代码编辑器中内容

(6) 将下列代码添加到图 6-8 所示的类中:

```
private int duration;
Image ii;//声明一个 Image 对象
int iconWidth=480;//欢迎窗口的宽度
int iconHeight=283;//欢迎窗口的高度
int liveTime,currTime;
public WelcomeWindow(int duration){
    liveTime=duration/1000;
    ii=this.getToolkit().getImage("welcome.JPG");//获取 image 对象
    //获取屏幕的大小以及欢迎窗口的 x 坐标和 y 坐标
    Dimension welcomeScreen=Toolkit.getDefaultToolkit().getScreenSize();
    int x=(welcomeScreen.width-iconWidth)/2;
    int y=(welcomeScreen.height-iconHeight)/2;
    currTime=liveTime;
    this.setBounds(x,y,iconWidth,iconHeight);设置窗口的大小和位置
    this.setVisible(true);
    this.setAlwaysOnTop(true);
}
public void showWelcomeWindow(){
    for(int z=0;z<liveTime;z++){
        try{
            currTime=liveTime-z;
            repaint();//重绘整个窗体
            Thread.sleep(1000);//睡眠 1000ms
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    this.setVisible(false);
}
public void showWelcomeWindowAndExit(){
    this.showWelcomeWindow();
    System.exit(0);
}
public void update(Graphics g){
    paint(g);
}
public void paint(Graphics g){
    g.drawImage(ii,0,0,this);//绘制图像
    Font font=new Font("宋体",Font.PLAIN,26);//设置字体
    g.setFont(font); //设置字体
    g.setColor(Color.white)设置颜色;
    g.drawString("该窗口在"+currTime+"秒钟后自动消失",80,150);//绘制字符串
}
```

- paint 方法用来绘制图像。

- showWelcomeWindow 方法调用 repaint 方法，并使线程睡眠 1000ms。
- showWelcomeWindowAndExit 调用 showWelcomeWindow 方法，然后退出虚拟机。

(7) 向 DemoJWindow 类添加一个 main() 方法，该方法的作用是创建 WelcomeWindow 对象，并调用其 showWelcomeWindowAndExit 方法。将下列代码添加到该方法中：

```
WelcomeWindow welcome=new WelcomeWindow(10000);//欢迎界面存在 10000ms  
welcome.showWelcomeWindowAndExit();
```

提示 在编辑代码的过程中，源代码编辑器会提示某些类没有被导入，按提示导入这些类即可。

(8) 在项目窗口中选中 DemoJWindow.java 节点，单击鼠标右键，在弹出的菜单中选择“Compile File”选项编译该文件。

(9) 编译成功后，选择右键菜单中的“Run File”选项，运行该程序，程序运行界面如图 6-9 所示。



图 6-9 程序运行界面

6.3 小应用程序——JApplet

Java 能够编写在浏览器中使用的程序，它能从网络上下载并运行。这种能在浏览器中使用的程序，就被称为 Applet。本章将为读者讲述 Swing 包中的 JApplet，主要包括如下内容：

- JApplet 简介；
- JApplet 的标记和属性；
- JApplet 的安全机制；
- JApplet 的生命周期；
- NetBeans 中 JApplet 的使用。

6.3.1 JApplet 简介

JApplet 简单地扩展了 java.applet.Applet，如果要在 Web 浏览器中使用 Swing 控件，则需要用到 JApplet。这里创建一个简单的 JApplet，代码设置如下：

```
JApplet myJApplet=new JApplet();
```

虽然 JApplet 提供了上面所用到的 JApplet 构造器，但是实际中开发人员很少会自己创建

JApplet 对象。通常情况下,都是构造一个扩展了 JApplet 的类,然后在一个 Web 页面中使用该类,由浏览器自动创建 JApplet 对象。下面介绍创建并使用 JApplet 的方法。

(1) 创建一个扩展了 JApplet 的类:

```
public class MyJApplet extends JApplet{//实现代码};
```

(2) 在一个 HTML 页面中使用 “<applet>” 标记来访问相应的 applet:

```
<applet code=" MyJApplet.class" width="500" height="400"></applet>
```

下面给出了一个在 JApplet 中显示 “Hello JApplet!” 的完整示例。按照如下步骤完成该例:

(1) 编写程序代码如下(文件名为 HelloApplet.java):

```
import javax.swing.*;
public class HelloApplet extends javax.swing.JApplet {
    public void init(){
        JPanel myJPanel=new JPanel();
        JLabel myJLabel=new JLabel("Hello JApplet!");
        myJPanel.add(myJLabel);
        this.add(myJPanel);
    }
}
```

(2) 编译源代码后,创建一个名称为 HelloApplet.html 的 HTML 文件,将下列代码添加到该文件中:

```
<applet code="HelloApple" height="400" width="500"></applet>
```

提示 虽然 HTML 的名字可以是任意的,但是通常情况下将其名称设置为与 JApplet 的类名一致。

(3) 在控制台输入 appletviewer HelloApplet.html,即出现如图 6-10 所示的运行界面。



图 6-10 Hello 示例

appletviewer 是 JDK 提供的一种查看 Applet 的工具,可以使用其代替浏览器查看包含 Applet 的网页。与浏览器不同的是,appletviewer 没有进行安全方面的限制。因此,对于安全性要求高的程序,则不使用 appletviewer 进行测试。

6.3.2 Applet 的标记和属性

在前面的小节中已经介绍,对于一个已经开发好的 JApplet,需要在 Web 页面通过<applet>

标记来使用该 JApplet。<applet>与</applet>标记是必须的，如果遗漏了其中的任何一个，那么浏览器不能加载 Applet。如果浏览器不支持使用 Applet，那么<applet>与</applet>之间的内容会被显示出来。对于这种浏览器来说，可以在两个标记间放入一些提示信息，以提示用户不能正常加载 Applet。

下面代码显示了如果浏览器不能使用 Applet，则会出现“你的浏览器不支持 Java”的提示信息。

```
<applet code="HelloApple" height="400" width="500">
你的浏览器不支持 Java
</applet>
```

使用“<applet>”标签时经常用到的属性及说明如表 6-6 所示。

表 6-6 applet 常用属性及说明

属 性	说 明
width	设置 Applet 的宽
height	设置 Applet 的高
align	设置 Applet 的对齐方式
code	设置要加载的 Applet 类的文件名
codebase	设置 Applet 类文件所在的目录
archive	用于 Applet 的 class 文件和其他资源的 jar 文件

表 6-7 applet 的对齐方式

属 性	说 明
left	将 Apple 放于页面的左边，页面中的文本显示在 Apple 的右边
right	将 Apple 放于页面的右边，页面中的文本显示在 Apple 的左边
top	把 Apple 的顶部对齐当前行的顶部
bottom	把 Applet 的底部对齐当前行文本的底部
texttop	把 Apple 的顶部对齐当前行的文本的顶部
middle	把 Apple 的中间对齐当前行的基线
absmiddle	把 Apple 的中间对齐当前行的中间
baseline	把 Apple 的底部对齐当前行的基线
absbottom	把 Apple 的底部对齐当前行的底部
vspace,hspace	分别指定了 Apple 的上下空白像素 (vspace) 和左右空白像素 (hspace)

以下代码显示了使用 left 作为 Applet 的对齐方式：

```
<applet code="HelloApple" height="400" align="left" width="500">
</applet>
```

6.3.3 Applet 的安全机制

因为 Applet 从远程站点下载后在本地执行，所以安全性很重要。如果用户在浏览器中启用了 Java，那么浏览器就会下载网页中的 Applet 代码并立即执行。用户永远也没有机会来确认或者停止 Applet 的运行。有鉴于此，Applet 在安全方面被作出了很多限制，当一个 Applet

试图突破这些限制时，就会抛出异常终止执行。

Applet 的限制执行环境通常被称为“沙箱 (sandbox)”，运行在“沙箱”中的 Applet 不能更改或探索用户的系统的敏感信息。当 Applet 运行在“沙箱”中时：

- 绝对不能启动运行任何本地可执行程序；
- 不能绑定到本地端口，以伪装成合法服务；
- 不能读写本地计算机的文件系统；
- 弹出的所有窗口都会带有警告信息。

“沙箱”机制能够成立的惟一原因是 Applet 是由虚拟机解释的，并不在用户计算机上直接运行。因为解释器会检查所有的危险指令和程序区域，所以一个危险的 Applet 不可能使计算机崩溃，覆盖系统内存或者改变操作系统的特权。

6.3.4 Applet 的生命周期

Applet 的生命周期主要包括 init、start、stop 和 destroy 4 个阶段。图 6-11 给出了这 4 种状态的转换情况。

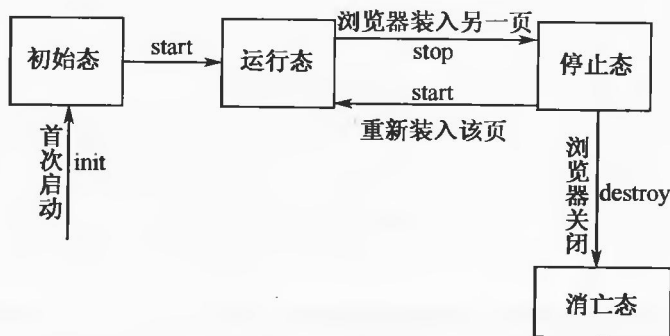


图 6-11 Applet 生命周期示意图

JApplet 类中的 init()、start()、stop()、destroy() 4 个方法可以在任意的 JApplet 中使用，下面是关于这些方法的简单说明，包括调用的时机和位置。

(1) init：该方法用于初始化 Applet，当 Applet 首次被加载时，该方法会自动被调用一次。

提示 JApplet 也可以有一个默认构造器，但是一般在 init 方法而非默认的构造器中进行初始化过程。

(2) start：该方法在 Java 调用 init 方法之后被调用，当用户从其他页面返回到包含 Applet 的页面中时，该方法也会被调用，它与 init 方法不同。有鉴于此，应该把调用一次的代码放到 init 方法中，而非 start 方法中。start 方法通常是 Applet 重起一个线程的地方，如果当用户回到当前网页的时候不需要做任何操作，那么就不需要实现此方法。

(3) stop：该方法在用户离开 Applet 页面时，被自动调用。因此，其也能被多次重复调用。主要作用是当用户不再关注 Applet 时能够停止一些没用的动作。典型作用是挂起一个线程。

(4) destroy：当浏览器被正常关闭时，JVM 会自动调用此方法。该方法中放置的代码主

要是用来回收一些系统资源。

6.3.5 NetBeans 中 JApplet 的使用

前面的章节对 JApplet 的使用方法以及特性做了一些简单的介绍。在实际的应用中，经常会将已有的 Java 应用程序转换到 Applet 中，以便能够在浏览器中应用该程序。将一个 Java 应用程序转化成 Applet，具体操作步骤如下。

(1) 创建一个扩展了 JApplet 的类。需要将其访问限制修饰符设置为 public，否则 Applet 不能被加载。

(2) 取消应用程序中的 main() 方法。

(3) 把应用程序中的所有初始化代码全部移到 Applet 的 init 方法中。不需要显式地创建 Applet 对象，浏览器在创建 Applet 对象时会自动调用 init 方法。

(4) 删除对 setSize 方法的调用。对于 Applet 来说，其大小是由实际 HTML 文件中的 width 和 height 属性值决定的。

(5) 删除对 setDefaultCloseOperation 的调用。当浏览器退出时，Applet 会被终止。

(6) 如果程序中调用了 setTitle 方法，则应该取消该调用。Applet 不能有标题栏（网页可以有标题栏，其值由 HTML 中的 title 标记来确定）。

(7) 不需要使用 setVisible() 方法，Applet 会自动显示。

(8) 创建一个 HTML 页，其中使用了正确的标记来加载 Applet 代码。

本节将通过一个例子介绍如何在 NetBeans 中使用 JApplet，按照如下步骤完成该例。

(1) 打开本章前面创建的项目 component，在项目窗口中选 org.netbeans.swing.component 节点，然后选择“File”主菜单下的“NewFile”选项。弹出如图 6-12 所示的窗口。

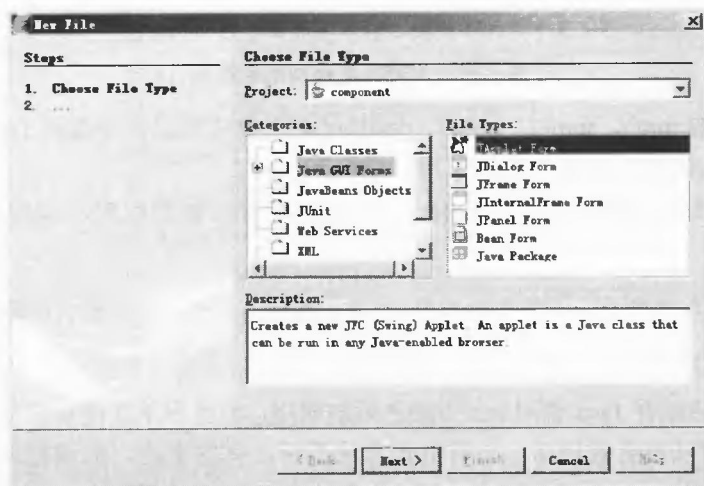


图 6-12 新建文件对话框

(2) 在“Categories”列表框中选择“Java GUI Form”选项，然后再选择“File Types”列表框中的“JApplet Form”选项，单击“Next”按钮，打开如图 6-13 所示的窗口。

(3) 在“Class Name”文本框中设置新添加类的类名，在此将其设置为 DemoJApplet。单击“Finish”按钮完成添加。

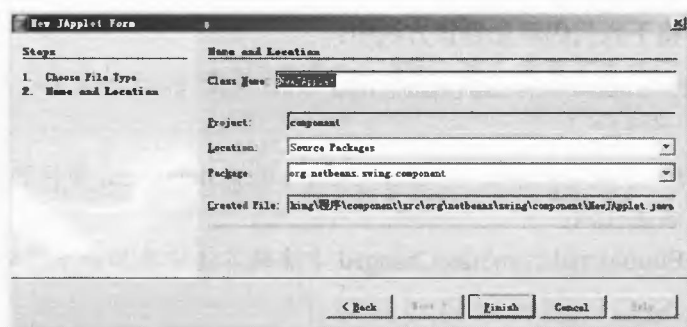


图 6-13 设置新添加文件的名称

(4) 在源代码编辑器中查看该类的源代码，可以发现 NetBeans 自动生成了 `init` 方法，方法中的代码如下：

```
try {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            initComponents();
        }
    });
} catch (Exception ex) {
    ex.printStackTrace();
}
```

(5) 打开 GUI 设计器，向 `DemoJApplet` 中添加一个 `JLabel`，具体的添加方法与向 `JFrame` 中添加 `JLabel` 的方法一致。将该 `JLabel` 的名称修改为 `jLabelFontMessage`，并将其 `text` 属性的值修改为 applet 应用示例。

(6) 向 `DemoJApplet` 中添加 2 个 `JCheckBox` 和 3 个 `JRadioButton`，按照如表 6-8 所示修改其属性。

表 6-8

属性与值对照表

控件类型	名称	text 属性值
<code>JCheckBox</code>	<code>jCheckBoxBold</code>	黑体
<code>JCheckBox</code>	<code>jCheckBoxItalic</code>	斜体
<code>JRadioButton</code>	<code>jRadioButtonSmall</code>	小字体
<code>JRadioButton</code>	<code>jRadioButtonCommon</code>	普通字体
<code>JRadioButton</code>	<code>jRadioButtonBig</code>	大字体

(7) 向 `DemoJApplet` 中添加一个 `ButtonGroup`，将其名称修改为 `buttonGroupOne`。然后将上面添加的 3 个 `JRadioButton` 的 `buttonGroup` 属性分别设置为 `buttonGroupOne`。

(8) 依次向 `DemoJApplet` 中添加名称分别为 `FontSize`、`style` 的两个成员变量。访问限制修饰符号均设置为 `private`，而返回值都设置为 `int`。将其初始值分别设置为 18、`java.awt.Font.PLAIN`。

(9) 为这两个成员变量编写访问器。

(10) 向 `DemoJApplet` 中添加名称为 `setLabelFont`，返回值为 `void`，访问限制修饰符号为

private 的方法，并将下列代码添加到该方法中：

```
this.jLabelFontMessage.setFont(new java.awt.Font("宋体",this.getStyle(),  
this.getFontSize()));
```

(11) 分别为每一个 JCheckBox 和 JRadioButton 添加 ItemEvent 事件的处理方法。并按照下列说明向方法中添加代码：

- 向 jRadioButtonSmallItemStateChanged 方法的方法中添加如下代码：

```
this.setFontSize(14);  
this.setLableFont();
```

- 向 jRadioButtonCommonItemStateChanged 方法的方法中添加如下代码：

```
this.setFontSize(18);  
this.setLableFont();
```

- 向 jRadioButtonBigItemStateChanged 方法中添加如下代码：

```
this.setFontSize(22);  
this.setLableFont();
```

- 向 jCheckBoxItalicItemStateChanged 方法中添加如下代码：

```
if(this.jCheckBoxItalic.isSelected()){  
    this.setStyle(java.awt.Font.ITALIC);  
}else {  
    this.setStyle(~java.awt.Font.ITALIC);  
} this.setLableFont();
```

- 向 jCheckBoxBoldItemStateChanged 方法中添加如下代码：

```
if(this.jCheckBoxBold.isSelected()){  
    this.setStyle(java.awt.Font.BOLD);  
}else{  
    this.setStyle(~java.awt.Font.BOLD);  
} this.setLableFont();
```

(12) 在项目窗口中选中 DemoJApplet.java 节点，单击鼠标右键，在弹出的菜单中依次选择“Run File”选项，则会弹出程序的运行窗口，如图 6-14 所示。

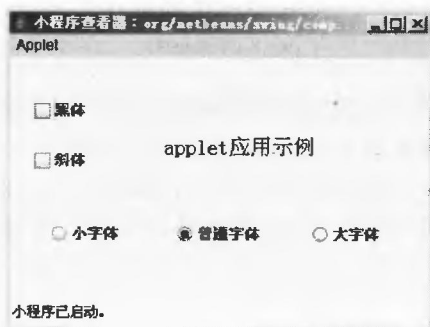


图 6-14 程序运行界面

说明 该程序的功能与 DemoJCheckRadioFrame 基本一致，这里不再演示。

提示 使用步骤（12）中所述方法运行 Applet 时，NetBeans 会自动生成相应的 HTML 页，并调用 appletviewer 查看 Applet，而不需要开发人员编写代码。

6.4 小结

本章介绍了 Java 中提供的一些容器类，包括 JFrame、JWindow 以及 JApplet，其中重点介绍了 Applet 的生命周期、工作机制等内容。同时还结合具体的项目详细地讲述了如何在 NetBeans 中使用这些容器类。可以看到在 NetBeans 中开发使用容器类的程序，是很简单的事情。

第 7 章

布局管理器

布局管理器是 Java 提供的一项非常重要的功能。通过使用布局管理器，开发人员可以更好地管理自己程序界面的布局。本章将详细地讲述布局管理，主要包括如下内容：

- 为什么使用布局管理器；
- Java 中的布局管理器；
- 各种布局的实现；
- 用 NetBeans 开发使用复合布局管理器的程序。

7.1 为什么使用布局管理器

在实际编程中，每设计一个窗体，都要往其中添加若干控件。将控件加入容器之后，应该按照一定的顺序和规则放置这些控件，使之看起来更美观，这就是布局。在 Java 中，布局由布局管理器（LayoutManager）来管理。

在 Windows 上，布局管理器不存在难以解决的问题的是对于 Windows 开发人员，可以将控件拖到窗体的表面上，然后通过编辑器工具来完成控件的对齐、均衡间距、中心定位等工作。这就使得布局管理器很重要。

但是，如果使用上述方法，当控件的大小发生改变时，则使用这种方法的布局就不允许手工更新。例如增大了按钮或标签上显示的字体，而按钮或标签并不会随之增大，所以增大的字体只能在原来的空间内。此时，就已经不能正常显示内容了。

Java 中的布局管理器是用于对控件进行布局的一个很好的工具。使用布局管理器以后，布局就会使用控件间关系的命令来完成，而不是手工设置。在最初的 AWT 中，这更显得重要，因为 AWT 使用本地图形元素来绘制控件，而 Motif、Windows 和 Macintosh 中的控件都不相同，并且程序也不会预先知道将要在哪个平台上运行（因为 Java 程序是要跨平台的）。

7.2 Java 中的布局管理器

Java 的 GUI 界面多数是使用 `java.awt` 包与 `java.swing` 包中定义类来开发的。其在布局管理上采用了容器和布局管理分离的方案。也就是说，容器只用来存放控件，而不管这些控件是如何放置的。对于布局的管理交给专门的布局管理器类来完成。

Java 中提供了很多布局管理器，表 7-1 给出了常用的布局管理器及其说明。

表 7-1 常用布局管理器及其说明

布局管理器	说 明
<code>java.awt.FlowLayout</code>	将控件按从左到右，从上到下的顺序依次排列，一行不能放完则折到下一行继续放置
<code>java.awt.BorderLayout</code>	将控件按东、南、西、北、中 5 个区域放置，每个区域最多只能放置一个控件
<code>java.awt.GridLayout</code>	形似一个无框线的表格，每个单元格中放一个控件
<code>java.awt.CardLayout</code>	将控件像卡片一样放置在容器中，在某一时刻只有一个控件可见
<code>Javax.swing.BoxLayout</code>	允许纵向或横向布置多个控件的布局管理器
<code>Javax.swing.SpringLayout</code>	根据一组约束条件放置子控件
<code>Javax.swing.OverlayLayout</code>	以彼此覆盖的形式叠置控件，要在彼此顶部安排控件的布局管理器。请求的容器大小将是请求的子容器的最大大小，同时要考虑对齐方式的需求。对齐方式以满足分配区域中子容器的需求为基础。放置子容器，使其对齐点都在彼此的顶部
<code>Javax.swing.ViewportLayout</code>	<code>JViewport</code> 的默认布局管理器，其定义了对大多数应用程序都很有用的布局策略。视口使其视图与视口的大小相同，但它不会使视图小于其最小大小。随着视口增大，视图先是在可以看到整个视图之前保持底部对齐，之后是顶部对齐
null 布局	使用硬性编码的方式管理控件的布局

表 7-1 仅仅给出了 Java 众多布局管理器中的一部分，本章后面的小节中将会对表 7-1 中给出的一些布局管理器进行详细的讨论。如果需要用到其他的布局管理器，可以查阅相关的 API。

7.3 流布局

流布局（`FlowLayout`）是初学者经常用到的一种布局，本节中将对该布局管理器进行讨论，主要包括如下内容：

- `FlowLayout` 简介；
- 在 NetBeans 中使用 `FlowLayout` 布局。

7.3.1 `FlowLayout` 简介

`FlowLayout` 是一种使用比较简单的布局管理器，其将控件按照从左到右、从上到下的顺序依次排列，一行不能放完则折到下一行继续放置。图 7-1 是使用了 `FlowLayout` 布局管理器

的程序运行界面。如果再向窗口中添加“按钮 7”，则窗口如图 7-2 所示。



图 7-1 使用 FlowLayout 的程序运行界面



图 7-2 添加“按钮 7”窗口布局

正如前面所述，由 FlowLayout 管理的布局，添加到容器中的每一行控件都被居中显示。如果已经到达行尾，则另起一行。使用鼠标拖动窗口的边框来对窗口进行缩放，仍然会遵循上面的显示规则，如图 7-3 所示。

下面为一个 JFrame 窗体设置 FlowLayout 布局管理器，代码如下：

```
JFrame frameLayout=new JFrame();
frameLayout.setLayout(new FlowLayout());
```

开发人员可以决定每一行上控件的排列，默认情况是居中显示，也可以在 FlowLayout 的构造器中指定对齐方式，代码如下：

```
frameLayout.setLayout(new FlowLayout(FlowLayout.LEFT));
```

FlowLayout 类有 3 个构造器，如表 7-2 所示。

表 7-2 FlowLayout 构造器及说明

构造器	说明
FlowLayout()	构造一个新的 FlowLayout，居中对齐，默认的水平 and 垂直间距是 5 个像素
FlowLayout(int align)	使用默认的水平 and 垂直间距和指定的对齐方式构造一个新的 FlowLayout。对齐方式的值必须是以下之一：FlowLayout.LEFT、FlowLayout.RIGHT、FlowLayout.CENTER、FlowLayout.LEADING 或 FlowLayout.TRAILING
FlowLayout(int align,int hgap, int vgap)	指定的对齐方式以及指定的水平和垂直间距构造新的 FlowLayout。align 参数表示对齐值，hgap 参数指定控件之间以及控件与容器的边框之间的水平间距，vgap 参数指定控件之间以及控件与容器的边之间的垂直间距

下面给出了一个使用 FlowLayout 布局管理器的完整示例代码：

```
package org.netbeans.swing.flowlayout;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class DemoFlowLayout extends JFrame implements ActionListener{
    JButton buttonAction;
    public DemoFlowLayout() {
        this.setLayout(new FlowLayout());
        buttonAction =new JButton("动作按钮");
        buttonAction.addActionListener(this);
    }
}
```



图 7-3 缩放窗口后的界面

```
this.add(buttonOne);
this.setBounds(150,150,350,120);
this.setTitle("FlowLayout 完整示例");
this.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
}
public void actionPerformed(ActionEvent e) {
    if(e.getSource()==buttonAction){
        JButton buttonCommon=new JButton("普通按钮");
        this.add(button);
        this.validate();
    }
}
public static void main(String args[]){
    new DemoFlowLayout().setVisible(true);
}
}
```

编译并运行程序，界面如图 7-4 所示。单击“动作按钮”按钮，可以向容器中添加“普通按钮”按钮，单击 9 次“普通按钮”按钮后，界面如图 7-5 所示。



图 7-4 程序运行界面



图 7-5 向容器中添加“普通按钮”

从上面的例子可以看到，使用 FlowLayout 进行布局管理，开发人员并不能够完全控制界面的布局，有时候可能会与要求相差很远。因此，对于复杂的界面，不应该单独使用 FlowLayout 进行布局管理。

7.3.2 在 NetBeans 中设置 FlowLayout 布局

在 NetBeans 中为容器设置 FlowLayout 布局非常简单。如果要为 JFrameFlow 的容器设置 FlowLayout 布局，首先在对象观察器窗口中的 JFrameFlow 节点上，单击鼠标右键，在弹出的菜单中依次选择“Set Layout”/“FlowLayout”选项，即可将 JFrameFlow 的布局管理器设置为 FlowLayout。

布局设置完成以后，如果向容器中添加控件，则控件会自动按照从左到右或从上到下的顺序进行排列。在对象观察器窗口中选中的 JFrameFlow 节点下面的 FlowLayout 节点，在属性窗口中可以对 FlowLayout 相应的属性进行设置。

7.4 边框布局

边框布局（BorderLayout）也是一种常用的布局管理器。本节将对该布局管理器进行介

绍, 主要包括如下内容:

- BorderLayout 简介;
- 在 NetBeans 中设置 BorderLayout 布局。

7.4.1 BorderLayout 简介

BorderLayout 是 JFrame 的默认布局管理器, 与 FlowLayout 那样完全由布局管理器控制每个控件的位置不同, BorderLayout 允许开发人员选择每个控件的摆放位置。BorderLayout 将容器划分为 5 个区域, 可以选择把控件放在容器的中间、北边、南边、东边或者西边, 如图 7-6 所示。

实际上, BorderLayout 布局管理器为 North 和 South 设置合适的高度, 而宽度设置为容器的宽度。为 East 和 West 设置合适的宽度, 而高度设置为容器的高度, Center 占据剩余的区域。当容器大小改变时, 边缘控件的厚度不会改变, 但是中间控件需要改变大小。改变如图 7-6 所示窗口的大小, 结果如图 7-7 所示。

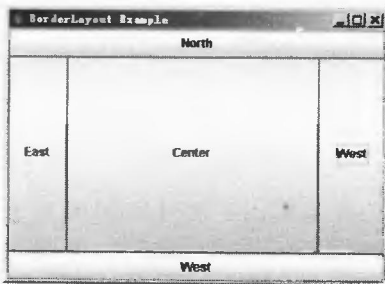


图 7-6 使用 BorderLayout 布局管理器

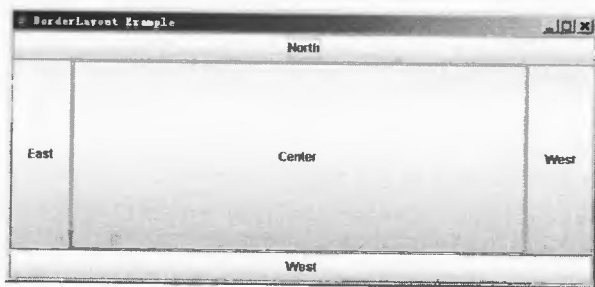


图 7-7 改变窗口大小效果图

可以通过指定 BorderLayout 类中的 CENTER、NORTH、SOUTH、EAST 或者 WEST 常量来向容器中指定位置添加控件。如果不提供任何值, 则系统就认为是 CENTER。

提示 BorderLayout 常量被定义为字符串。例如, BorderLayout.SOUTH 被定义为字符串 South。许多开发人员更愿意直接使用字符串, 例如 getContentPane.add(component, "South")。然而, 如果拼写错误, 编译器不会报错。

以下代码为向一个 JFrame 的 North 区域添加一个按钮:

```
JFrame borderFrame=new JFrame();
JButton northButton=new JButton();
borderFrame.add(northButton, BorderLayout.NORTH);
```

默认情况下, BorderLayout 的各个区域之间是没有空隙的, 可以使用 BorderLayout 类提供的 BorderLayout(int hgap, int vgap) 构造器创建一个各区域之间有间距的 BorderLayout 布局管理器。这两个参数用来设置各个区域之间的水平与垂直空隙。

如果某个区域没有添加任何控件, 则该区域的大小将变为零, 其他区域将会扩展并占用

这些未使用的区域。没有 North 区域的使用 BorderLayout 布局管理器的窗口如图 7-9 所示。

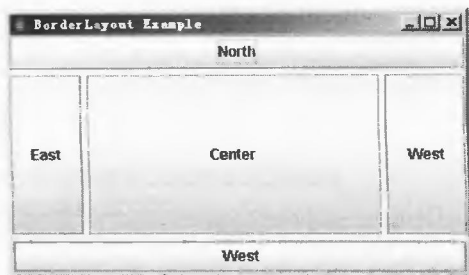


图 7-8 区域之间有间距的窗口

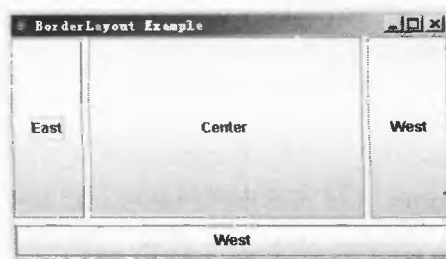


图 7-9 没有 North 区域的窗口

下面给出了使用 BorderLayout 布局管理器设置布局的完整示例代码：

```
package org.netbeans.swing.borderlayout;
import java.awt.*;
import javax.swing.*;
public class DemoBorderLayout extends JFrame{
    /*
     * 声明一些控件
     */
    private JLabel jLabelEast, jLabelWest, jLabelNorth, jLabelCenter;
    private JPanel jPanelSouth;
    private JButton jButtonOne, jButtonTwo, jButtonThree;
    public DemoBorderLayout() {
        // 创建一个水平和垂直间距都为 5 的 BorderLayout
        BorderLayout myBorderLayout = new BorderLayout(5, 5);
        // 设置 DemoBorderLayout 的布局管理器
        this.setLayout(myBorderLayout);
        // 创建具有指定内容的 JLabel 对象
        jLabelEast = new JLabel("这是 East 区域");
        jLabelWest = new JLabel("这是 West 区域");
        jLabelNorth = new JLabel("这是 North 区域");
        jLabelCenter = new JLabel("这是 Center 区域");
        // 设置 JLabel 中文本的对齐方式
        jLabelEast.setHorizontalAlignment(SwingConstants.CENTER);
        jLabelWest.setHorizontalAlignment(SwingConstants.CENTER);
        jLabelNorth.setHorizontalAlignment(SwingConstants.CENTER);
        jLabelCenter.setHorizontalAlignment(SwingConstants.CENTER);
        // 创建具有指定内容的 JButton 对象
        jButtonOne = new JButton("ButtonOne");
        jButtonTwo = new JButton("ButtonTwo");
        jButtonThree = new JButton("ButtonThree");
        // 创建 JPanel 对象，将其布局管理器设置为 FlowLayout
        jPanelSouth = new JPanel();
        jPanelSouth.setLayout(new FlowLayout());
        // 向 jPanelSouth 的不同区域中添加按钮
```

```
jPanelSouth.add(jButtonOne);
jPanelSouth.add(jButtonTwo);
jPanelSouth.add(jButtonThree);
//将主窗口的各个不同的区域中添加相应的控件
this.add(jLabelEast,BorderLayout.EAST);
this.add(jLabelWest,BorderLayout.WEST);
this.add(jLabelNorth,BorderLayout.NORTH);
this.add(jLabelCenter,BorderLayout.CENTER);
this.add(jPanelSouth,BorderLayout.SOUTH);
this.setBounds(150,150,450,220);//设置主窗口的大小
this.setTitle("BorderLayout 应用完整示例");//设置主窗口的标题
this.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
public static void main(String args[]) {
    new DemoBorderLayout().setVisible(true);
}
}
```

编译并运行程序，界面如图 7-10 所示。

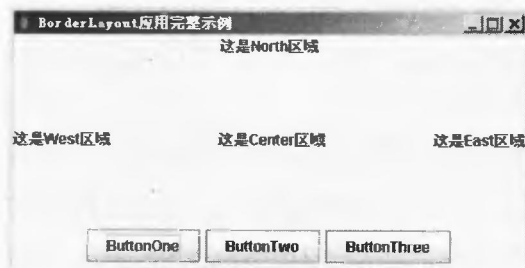


图 7-10 程序运行界面

7.4.2 在 NetBeans 中设置 BorderLayout 布局

在 NetBeans 中为容器设置 BorderLayout 布局管理器非常便捷。如果为名称为 `jFrameBorder` 的容器设置 BorderLayout 布局，首先在 `jFrameBorder` 节点上单击鼠标右键，然后依次选择菜单中的“Set Layout”/“BorderLayout”选项，即可将 `jFrameFlow` 的布局管理器设置为 BorderLayout。

布局管理器设置完成以后，即可向容器中添加控件。在“Palette”面板中选择一个控件，把鼠标移动到 GUI 设计器中，GUI 设计器会提示当前控件所处的区域，将鼠标移动到将要添加控件的区域，单击鼠标即可完成添加，如图 7-11 所示。



每个区域内只能添加一个控件，如果向同一区域内先后添加一个以上的控件，则后添加的控件不可见。

在对象观察器窗口中选 `jFrameBorder` 节点下面的 BorderLayout 节点，在属性设置窗口中通过修改 Horizontal Gap 属性与 Vertical Gap 属性的值可以设置不同区域间的水平间距与垂直

直间距。

7.5 网格布局

网格布局（GridLayout）是一种非常实用的布局管理器。使用 GridLayout 布局管理器，可以使开发类的程序变得更加简单。本节将讨论 GridLayout 布局管理器的使用，主要包括如下内容：

- GridLayout 简介；
- 在 NetBeans 中设置 GridLayout 布局。

7.5.1 GridLayout 简介

GridLayout 布局又称为网格布局。网格布局按照给定行数和列数排列所有控件，添加到具有网格布局的容器中的控件具有相同的大小。创建 GridLayout 布局管理器的对象，需要指定网格的行数和列数，代码如下：

```
JPanel gridPanel=new JPanel();  
gridPanel.setLayout(new GridLayout(4,3));
```

同 BorderLayout 和 FlowLayout 一样，也可以在 GridLayout 布局管理器中指定容器控件间的水平和垂直间距，代码如下：

```
gridPanel.setLayout(new GridLayout(4,3,3,3));
```

创建 GridLayout 对象以后，还可以使用 GridLayout 类中的一些方法对相应的属性进行设置，表 7-3 给出了 GridLayout 类中常用的方法及其说明。

表 7-3 BorderLayout 类的常用方法及其说明

方 法	说 明
getRows()	获取此布局中的行数
setRows(int rows)	将此布局中的行数设置为指定值
getColumns()	获取此布局中的列数
setColumns(int cols)	将此布局中的列数设置为指定值
getHgap()	获取控件之间的水平间距
setHgap(int hgap)	将组控件之间的水平间距设置为指定值
getVgap()	获取控件之间的垂直间距
setVgap(int vgap)	将控件之间的垂直间距设置为指定值

向 GridLayout 布局的容器中添加控件时，并不会将每一行中的网格都填满以后才填充下一行，或者将每一列填满后才填充下一行。例如，向一个指定了 3 行 2 列的容器中添加两个按钮，界面如图 7-12 所示。

再向容器中添加一个名称为“按钮 3”的按钮，界面如图 7-13 所示。继续向容器中添加一个名称为“按钮 4”的按钮，界面如图 7-14 所示。

可以看到，向 GridLayout 布局的容器中添加控件时，确实没有先把行/列中的每一个网格

填满以后才去填充下一行/列中的网格。

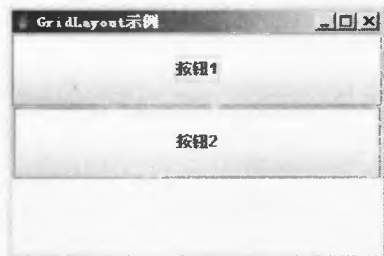


图 7-12 GridLayout 布局示例



图 7-13 添加“按钮 3”后界面

其规则是如果行和列的设置值都不为 0，则在保证行数的情况下使添加的控件尽量组成矩形，列数由行数和添加控件的总数来决定；如果行的值设置为 0 而列值不为 0，则在保证列数的情况下使添加的控件尽量组成矩形，行数由列数和添加控件的总数来决定。

例如向如图 7-13 所示的容器中添加 9 个按钮，其将在保证行数的情况下让控件尽量组成矩形，所以为 3×3（在行数不为 0 时列不起作用），如图 7-15 所示。

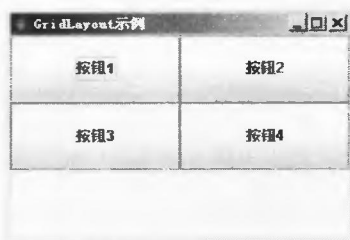


图 7-14 添加“按钮 4”后界面



图 7-15 向 3 行 2 列的容器中添加 9 个按钮

从上面的几副效果图中可以看出，GridLayout 也不是一种很完美的布局。另外，当用户改变容器的大小时，为了达到充满整个容器的效果，所有的控件都将被改变大小。

7.5.2 在 NetBeans 中设置 GridLayout 布局

在 NetBeans 中将容器的布局管理器设置为 GridLayout 是非常简单的，本节将通过一个例子介绍如何在 NetBeans 中使用 GridLayout 布局。

创建一个名称为 layoutmanager 的项目，将其主类名称设置为 org.netbeans.swing.layoutmanager.DemoLayout，然后按照如下步骤完成该例。

(1) 向 layoutmanager 项目添加一个通过 JFrame Form 模版创建的 DemoGridLayout-Frame 类。

(2) 打开 GUI 设计器后，在对象观察器窗口中选中 JFrame 节点，将其布局管理器设置为 BorderLayout。

(3) 向面板的 West 区域中添加一个 JList，将其名称修改为 jListWest，并将其初始选项设置为 3×2、3×3、3×4、3×5、4×3。

(4) 向面板的 South 区域中添加一个 JPanel，将其名称修改为 jPanelSouth。JPanel 默认的布局管理器为 FlowLayout。向 jPanelSouth 中添加两个按钮，分别将这两个按钮的名称修改为 jButtonFlow 和 jButtonGrid，并将其 text 属性的值修改为“流布局”、“网格布局”。

(5) 向面板的 Center 区域中添加一个 JPanel，将其名称修改为 jPanelCenter。在对象观察器窗口中选中 jPanelCenter 节点，单击鼠标右键，在弹出的菜单中依次选择“Set Layout”/“GridLayout”选项，即可将 jPanelCenter 的布局管理器设置为 GridLayout。

(6) 用鼠标右键单击 jPanelCenter 节点下的 GridLayout 节点，选择右键菜单中的“Properties”选项，在弹出的属性对话框中将其 Columns 属性的值修改为 3，将 Rows 属性的值修改为 4。

(7) 依次向 jPanelCenter 中添加 11 个 JButton，分别将其名称设置为 jButtonOne、jButtonTwo……jButtonEleven。并依次将其 text 属性修改为“按钮 1”、“按钮 2”……“按钮 11”。

(8) 此时，界面如图 7-16 所示。

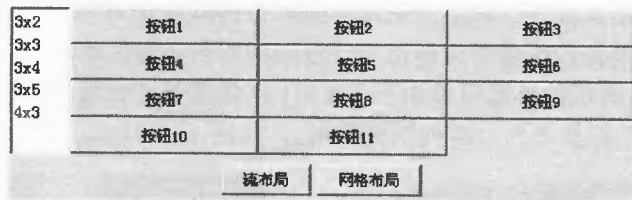


图 7-16 程序界面

(9) 为 jButtonFlow 与 jButtonGrid 添加(ActionEvent) 事件处理方法。为 jListWest 添加 ListSelectionEvent 事件处理方法。并向相应的事件处理方法内添加代码。

- 将下列代码添加到 jButtonFlowActionPerformed 方法中：

```
this.jPanelCenter.setLayout(new FlowLayout());
this.jPanelCenter.validate();
```

- 将下列代码添加到 jButtonGridActionPerformed 方法中：

```
this.jPanelCenter.setLayout(new GridLayout(4,3));
this.jPanelCenter.validate();
```

- 将下列代码添加到 jListWestValueChanged 方法中：

```
int rowColumns=this.jListWest.getSelectedIndex();
switch(rowColumns){
    case 0:
        this.jPanelCenter.setLayout(new GridLayout(2,3));
        this.jPanelCenter.validate();
        break;
    case 1:
        this.jPanelCenter.setLayout(new GridLayout(3,3));
        this.jPanelCenter.validate();
        break;
    case 2:
        this.jPanelCenter.setLayout(new GridLayout(3,4));
        this.jPanelCenter.validate();
        break;
```

```

        case 3:
            this.jPanelCenter.setLayout(new GridLayout(4,3));
            this.jPanelCenter.validate();
            break;
        case 4:
            this.jPanelCenter.setLayout(new GridLayout(5,3));
            this.jPanelCenter.validate();
            break;
    }
}

```

(10) 将下列代码添加到 DemoLayout 类的 public static void main(String args[])方法中:

```

java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoGridLayoutFrame().setVisible(true);
    }
});

```

(11) 编译并运行主项目, 如图 7-17 所示。

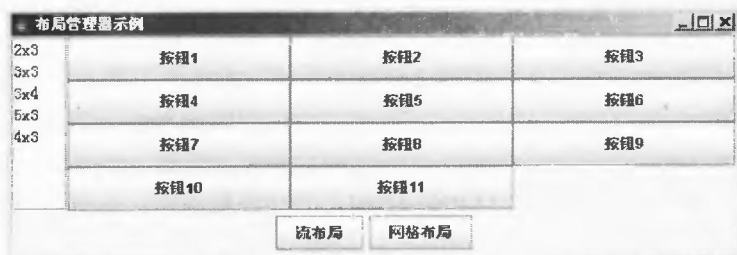


图 7-17 程序运行界面

(12) 单击“流布局”按钮, 则 jPanelCenter 使用 FlowLayout 布局管理器, 窗口如图 7-18 所示。



图 7-18 单击“流布局”按钮后的效果图

在如图 7-16 所示的状态下单击“网格布局”按钮, 则窗口如图 7-16 所示。

(13) 单击列表中的“2×3”选项, 窗口如图 7-19 所示。



说明

从图 7-17 中可以看出, 虽然将 jPanelCenter 的 BorderLayout 设置为 2 行 3 列, 但是实际上列值被忽略了。



图 7-19 选择“2×3”选项后的效果图

(14) 选择列表中的“5×3”选项，窗口如图 7-20 所示。

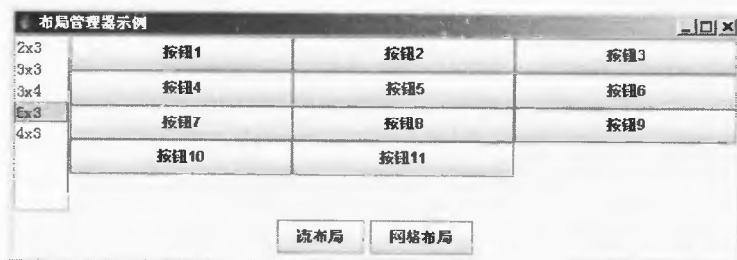


图 7-20 选择“5×3”选项后的效果图

7.6 卡片布局

Java 提供了一种称为卡片布局 (CardLayout) 的布局管理器，本节将该布局管理器进行介绍，主要包括如下内容：

- CardLayout 简介；
- 在 NetBeans 中使用 CardLayout 布局。

7.6.1 CardLayout 简介

CardLayout 是一种卡片式的布局管理器，其将容器中的控件看做是一系列的卡片。在一个显示区域内可以加入多张卡片，但每次只有一张卡片是可见的，就像一副扑克牌（在英语中 card 也有扑克牌的意思）。

下图中的例子使用两张卡片，一张卡片中是一个按钮控件，另一张是复选框控件，通过单击“Card One”或“Card Two”单选按钮可以在两张不同的卡片之间进行切换。分别单击两个单选按钮后的窗口如图 7-21 和图 7-22 所示。



图 7-21 选择“Card One”单选按钮



图 7-22 选择“Card Two”单选按钮

在任何时候, 容器中只有一张卡片是可见的, 可以通过如下方式选择将要被显示的“卡片”。

- 按“卡片”添加到容器中的顺序, 可以请求显示第一张或最后一张“卡片”。
- 显示当前卡片的前一张后或后一张“卡片”。
- 显示特定名称的“卡片”。

向一个由 `CardLayout` 布局管理器管理的容器中添加控件时, 可以指定一个能够标识该控件的字符串, 下面代码显示了如何创建使用 `CardLayout` 布局管理器的容器, 并向其中添加“卡片”:

```
JPanel cards=new JPanel(new CardLayout());
JPanel card1=new JPanel();
JPanel card2=new JPanel();
cards.add(card1,"firstcard");
cards.add(card2,"secondcard");
```

要显示由 `CardLayout` 所管理的容器中的控件, 需要使用一些额外的代码, 下面给出了如何显示 `card2` 的代码:

```
CardLayout cl=(CardLayout)(cards.getLayout());
cl.show(cards, "secondcard");
```

以上代码说明, 使用 `CardLayout` 类的 `show` 方法来设置当前显示的卡片。`show` 方法的第一个参数是 `CardLayout` 管理的容器, 第二个参数是标识要显示卡片的字符串, 该字符串与将控件添加到容器时使用的字符串相同。

另外, `CardLayout` 类还提供了其他几个方法来设置当前显示的卡片, 表 7-4 给出了这些方法及其说明。

表 7-4 `CardLayout` 的常用方法及其说明

方 法	说 明
<code>first(Container con)</code>	显示 <code>Container</code> 容器中的第一张卡片
<code>next(Container con)</code>	显示 <code>Container</code> 容器中的当前卡片的下一张卡片
<code>previous(Container con)</code>	显示 <code>Container</code> 容器中的当前卡片的前一张卡片
<code>last(Container con)</code>	显示 <code>Container</code> 容器中的最后一张卡片

提示 这 4 个方法的入口参数都是被布局管理器管理的容器。

下面给出了图 7-21 所示程序的完整代码:

```
package org.netbeans.swing.cardlayout;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class DemoCardLayout extends JFrame implements ItemListener {
    //声明控件
    JRadioButton jRadioButtonCardOne,jRadioButtonCardTwo;
    JButton jButtonOne;
```

```

JCheckBox jCheckBoxSecond;
JPanel jPanelNorth;
JPanel jPanelCenterFirst, jPanelCenterSecond, jPanelCenterBottom;
ButtonGroup buttonGroupOne=new ButtonGroup();;
public DemoCardLayout() {
    //创建相应的控件对象
    jPanelNorth=new JPanel();
    jPanelCenterFirst=new JPanel();
    jPanelCenterSecond=new JPanel();
    jPanelCenterBottom=new JPanel(new CardLayout());
    jPanelCenterFirst.setBackground(Color.GRAY);
    jPanelCenterSecond.setBackground(Color.white);
    jButtonOne=new JButton("我是卡片 1");
    jCheckBoxSecond=new JCheckBox("我是卡片 2",true);
    jRadioButtonCardOne=new JRadioButton("Card One");
    jRadioButtonCardTwo=new JRadioButton("Card Two");
    jRadioButtonCardOne.setSelected(true);设置单选按钮的选中状态
    //将两个单选按钮编组
    buttonGroupOne.add(jRadioButtonCardOne);
    buttonGroupOne.add(jRadioButtonCardTwo);
    //向单选按钮添加事件监听器
    jRadioButtonCardOne.addItemListener(this);
    jRadioButtonCardTwo.addItemListener(this);
    jPanelNorth.add(jRadioButtonCardOne);
    jPanelNorth.add(jRadioButtonCardTwo);
    jPanelCenterFirst.add(jButtonOne);
    jPanelCenterSecond.add(jCheckBoxSecond, BorderLayout.CENTER);
    //将 jPanelCenterFirst 添加到 jPanelCenterBottom 中, 并以 firstcard 标识
    jPanelCenterBottom.add(jPanelCenterFirst, "firstcard");
    //将 jPanelCenterSecond 添加到 jPanelCenterBottom 中, 并以 secondcard 标识
    jPanelCenterBottom.add(jPanelCenterSecond, "secondcard");
    this.add(jPanelNorth, BorderLayout.NORTH);
    将 jPanelCenterBottom 添加到主窗体的 CENTER 区域中
    this.add(jPanelCenterBottom, BorderLayout.CENTER);
    this.setBounds(150,150,450,220);
    this.setTitle("CardLayout 应用示例");
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
public void itemStateChanged(ItemEvent e){
    if(e.getSource()==jRadioButtonCardOne){
        //获取 jPanelCenterBottom 的布局管理器并将其转换为 CardLayout
        CardLayout clFirst=(CardLayout)jPanelCenterBottom.
getLayout();
        clFirst.show(jPanelCenterBottom, "firstcard");
    }
    if(e.getSource()==jRadioButtonCardTwo){
        //获取 jPanelCenterBottom 的布局管理器并将其转换为 CardLayout

```

```
        CardLayout clSecond=(CardLayout)jPanelCenterBottom.  
getLayout();  
        clSecond.show(jPanelCenterBottom,"secondcard");  
    }  
}  
public static void main(String args[]){  
    new DemoCardLayout().setVisible(true);  
}  
}
```

7.6.2 在 NetBeans 中使用 CardLayout 布局

在 NetBeans 中可以很方便地将容器的布局管理器设置为 CardLayout, 本节将通过一个例子, 介绍如何在 NetBeans 中进行 CardLayout 布局管理器的设置。打开前面创建的 layoutmanager 项目, 按照如下步骤完成该例。

(1) 向 layoutmanager 项目中添加一个通过 JFram Form 模版创建的名称为 DemoCardLayout 的类。并将其布局管理器设置为 BorderLayout。

(2) 分别向 DemoCardLayout 的 North 区域与 Center 区域中添加一个 JPanel, 将其名称分别修改为 JPanelNorth 和 JPanelCenter。

(3) 在对象观察器窗口中选中 JPanelCenter 节点, 单击鼠标右键, 在弹出的菜单中依次选择“Set Layout”/“CardLayout”选项, 即可将 JPanelCenter 的布局管理器设置为 CardLayout。

(4) 在对象观察器窗口中用鼠标右键单击 JPanelCenter 节点, 依次选择右键菜单中的“Add From Palette”/“Swing”/“JPanel”选项, 即向 JPanelCenter 中添加一个 JPanel, 将其名称修改为 JPanelCenterFirst, 并将其 Card Name 属性修改为 cardfirst。

(5) 按步骤 (4) 所述, 再向 JPanelCenter 添加一个名称为 JPanelCenterSecond 的 JPanel。并将其 Card Name 属性修改为 cardsecond。

Card Name 属性是把控件添加到容器中的标识字符串, 在以后可以与卡片布局管理器
提示 的 show 方法配合使用, 在本例中充当两张卡片的控件分别是 JPanelCenterFirst 和 JPanelCenterSecondl。

(6) 在对象观察器中选中 JPanelCenterFirst 节点, 此时, GUI 设计器中该卡片是可见的, 可以通过鼠标拖曳的方式向其中添加控件。向其中添加一个 JButton, 将其名称修改为 jButtonFirst, text 属性值修改为“我是卡片 1”。

(7) 按照步骤 (6) 所述向 JPanelCenterSecond 中添加一个 JCheckBox, 将其名称分别修改为 jCheckBoxSecond。并将其 text 属性值修改为“我是卡片 2”。

(8) 向 JPanelNorth 中添加两个名称为 jRadioButtonOne 和 jRadioButtonTwo 的单选按钮, 分别将其 Text 属性的值修改为 Card One 和 Card Two, 并将 jRadioButtonOne 的 select 属性设置为选中状态。

(9) 向 JPanelNorth 中添加一个 ButtonGroup, 将其名称修改为 buttonGroupOne。并将 jRadioButtonOne 和 jRadioButtonTwo 的 buttonGroup 属性均设置为 buttonGroupOne。

(10) 分别为 jRadioButtonOne 与 jRadioButtonTwo 添加 ItemEvent 事件的处理方法, 按照如下所述, 向相应的事件处理方法内添加代码。

- 向 `JRadioButtonOneItemStateChanged` 方法中添加如下代码:

```
CardLayout cardFirst=(CardLayout)jPanelCenter.getLayout();  
cardFirst.show(jPanelCenter,"cardfirst");
```

- 向 `JRadioButtonTwoItemStateChanged` 方法中添加如下代码:

```
CardLayout cardSecond=(CardLayout)jPanelCenter.getLayout();  
cardSecond.show(jPanelCenter,"cardsecond");
```

(11) 将下列代码添加到 `DemoLayout` 的 `public static void main(String args[])` 方法中:

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoCardLayout().setVisible(true);  
    }  
});
```

(12) 编译并运行程序, 运行结果与图 7-21 所示一致。

7.7 箱式布局

箱式布局 (`BoxLayout`) 是一种功能非常强大的布局管理器, 其将容器中的控件像堆箱子一样堆成一行或者一列。本节将对 `BoxLayout` 展开讨论, 主要包括如下内容:

- `BoxLayout` 简介;
- 使用 `Box` 类;
- 在 NetBeans 中使用 `BoxLayout` 布局。

7.7.1 BoxLayout 简介

Swing 包中提供了一种称为 `BoxLayout` 的布局管理器。`BoxLayout` 可以将容器中的控件放置成一行或者一列, 可以将其理解为 `FlowLayout` 的完全功能版本。使用 `BoxLayout` 布局管理器的程序的运行界面如图 7-23 所示。

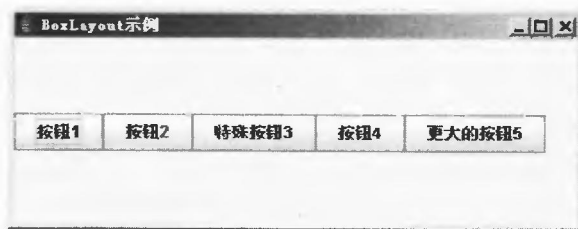


图 7-23 使用 `BoxLayout` 布局管理器

容器设置 `BoxLayout` 布局管理器, 代码如下:

```
Container contentPane=getContentPane();  
contentPane.setLayout(new BoxLayout(con, BoxLayout.X_AXIS));
```

`BoxLayout(Container target, int axis)` 是 `BoxLayout` 类的惟一构造器, `target` 参数指定了需要设置 `BoxLayout` 布局管理器的容器, `axis` 指定了控件在容器的摆放顺序, 其值在 `BoxLayout`

类中定义，表 7-5 给出了这些值并做出了相应的说明。

表 7-5 BoxLayout 中定义的显示方式

值	说 明
X_AXIS	从左到右横向布置控件
Y_AXIS	从上到下纵向布置控件
LINE_AXIS	指定应该根据目标容器的 ComponentOrientation 属性确定的文本行方向放置控件
PAGE_AXIS	指定应该根据目标容器的 ComponentOrientation 属性确定的文本行在页面中的流向来放置控件

图 7-22 所示的容器是使用 X_AXIS 作为其对齐方式，现将其对齐方式修改为 Y_AXIS 后，如图 7-24 所示。

BoxLayout 布局管理器总是试图按照控件的首选宽度（横向布局）或首选高度（纵向布局）来排列它们。对于横向布局，如果并不是所有的控件都具有相同的高度，则 BoxLayout 会试图让所有控件都具有最高控件的高度。

对于某些特定控件而言，如果不能达到最高控件的高度，则 BoxLayout 会根据该控件 Y 调整值对其进行纵向调整。默认情况下，控件的 Y 调整值为 0.5，这意味着控件的纵向中心应该与其他 Y 调整值为 0.5 的组件的纵向中心具有相同 y 坐标。

同样，对于纵向布局，BoxLayout 布局管理器总是试图让列中的所有控件具有最宽控件的宽度。如果这样做失败，则 BoxLayout 会根据这些控件的 X 调整值对它进行横向调整。

对于 PAGE_AXIS 布局，基于控件的开始边横向对齐。换句话说，如果容器的 Component Orientation 表示从左到右，则 X 调整值为 0.0 意味着控件的左边缘对齐，否则它意味着控件的右边缘对齐。

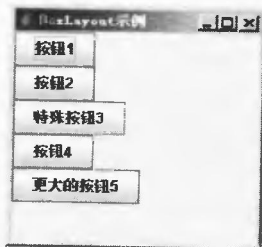


图 7-24 指定 Y_AXIS 的对齐方式

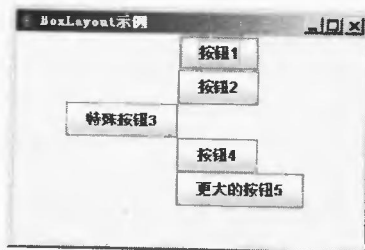


图 7-25 X 调整值为 1.0

BoxLayout 中提供了一些方法供开发人员使用，如表 7-6 所示。

表 7-6 BoxLayout 的常用方法及说明

方 法	说 明
preferredLayoutSize(Container target)	返回此布局的首选尺寸
minimumLayoutSize(Container target)	返回包含在指定目标容器中的控件所需的最小尺寸
maximumLayoutSize(Container target)	返回目标容器中所包含控件可使用的最大尺寸
getLayoutAlignmentX(Container target)	返回容器沿 x 轴的对齐方式。如果 box 是横向的，则返回默认对齐方式。否则，将返回沿 x 轴放置子控件所需的对齐方式
getLayoutAlignmentY(Container target)	返回容器沿 y 轴的对齐方式。如果 box 是纵向的，则返回默认对齐方式。否则，将返回沿 y 轴放置子控件所需的对齐方式

提示 这几个方法的入口参数都为被 `BoxLayout` 布局管理器管理的容器。

下面给出了该段程序的完整代码：

```
package org.netbeans.swing.boxlayout;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class DemoBoxLayout extends JFrame {
    JButton jButtonOne, jButtonTwo, jButtonThree, jButtonFour, jButtonFive;
    public DemoBoxLayout() {
        Container con=this.getContentPane();
        //将容器的布局管理器设置为 BoxLayout, 对齐方式为 Y_AXIS。
        con.setLayout(new BoxLayout(con, BoxLayout.Y_AXIS));
        jButtonOne=new JButton("按钮 1");
        jButtonTwo=new JButton("按钮 2");
        jButtonThree=new JButton("特殊按钮 3");
        //将 X 的调整值设置为 RIGHT_ALIGNMENT。
        jButtonThree.setAlignmentX(Component.RIGHT_ALIGNMENT);
        jButtonFour=new JButton("按钮 4");
        jButtonFive=new JButton("更大的按钮 5");
        con.add(jButtonOne); con.add(jButtonTwo);
        con.add(jButtonThree); con.add(jButtonFour);
        con.add(jButtonFive);
        this.setTitle("BoxLayout 示例");
        this.setBounds(100,100,300,200);
        this.setVisible(true);
        this.validate();
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[]) {
        DemoBoxLayout dlayout=new DemoBoxLayout();
    }
}
```

7.7.2 使用 Box 类

前面简单介绍了 `BoxLayout` 布局管理器的使用。可以看到，使用 `BoxLayout` 布局管理器的控件都与其邻近的控件“接壤”，这并不能满足所有控件的需要。可以向容器中添加一些不可视控件（`Box` 类），这样就可以使可视控件在界面中不是一个紧跟另一个。

`Box` 类直接扩展了 `java.swing.JComponent`，是使用 `BoxLayout` 的轻量级容器。`Box(int axis)` 是 `Box` 类惟一的构造器，其用来创建一个沿指定方向显示的 `Box` 容器对象。

`Box` 类可以创建几种影响布局的不可见控件，如 `glue`、`struts` 和 `rigid`。如果 `Box` 包含的所有控件都有一个固定大小，可以使用 `glue` 控件来控制可视控件的位置。如果想让两个控件

之间有一个固定的空间量, 可以尝试使用 `strut` 控件。如果需要一个总是占用相同空间量的不可见控件, 可以通过 `rigid` 来实现该功能。

以下代码显示了为放置在从左到右的 `BoxLayout` 中的两个控件之间设置宽度为 5 个像素的空间:

```
contentPane.add(firstComponent);
contentPane.add(Box.createRigidArea(new Dimension(5,0)));
contentPane.add(secondComponent);
```

`Box` 类提供了一些帮助开发人员更好地使用 `BoxLayout` 的便利方法, 表 7-7 给出了其中常用的方法及说明。

表 7-7 `BoxLayout` 的常用方法及说明

方 法	说 明
<code>createHorizontalBox()</code>	创建一个从左到右显示其控件的 <code>box</code>
<code>createVerticalBox()</code>	创建一个从上到下显示其控件的 <code>box</code>
<code>createRigidArea(Dimension d)</code>	创建一个总是具有指定大小的不可见控件
<code>createHorizontalStrut(int width)</code>	创建一个不可见的、固定宽度的控件。在一个横向 <code>box</code> 中, 通常使用此方法强行使两个控件之间具有某一空间量。在一个纵向 <code>box</code> 中, 可以使用此方法强行使 <code>box</code> 至少具有指定宽度。只有存在可用的剩余空间时, 不可见组件才有高度
<code>createVerticalStrut(int height)</code>	创建一个不可见的, 具有固定高度的控件。在一个纵向 <code>box</code> 中, 通常使用此方法强行使两个控件之间具有某一空间量。在一个横向 <code>box</code> 中, 可以使用此方法强行使 <code>box</code> 至少具有指定高度。只有具有可用的剩余空间时, 不可见控件才有宽度
<code>createGlue()</code>	创建一个不可见的 <code>glue</code> 控件, 经常用在其可见组件有一个最大宽度 (横向 <code>box</code>) 或高度 (纵向 <code>box</code>) 的 <code>box</code> 中。可将该 <code>glue</code> 组件视为一种粘性物质, 它尽其所需地进行扩张, 以填充与其相邻组件间的空间
<code>createHorizontalGlue()</code>	创建一个横向 <code>glue</code> 组件
<code>createVerticalGlue()</code>	创建一个纵向 <code>glue</code> 组件

提示 `Box` 类也提供了 `setLayout(LayoutManager)` 方法, 但是如果 `Box` 对象调用该方法, 则会抛出 `AWTError`, 因为 `Box` 只能使用 `BoxLayout` 布局管理器。

下面给出了使用 `Box` 的完整示例:

```
package org.netbeans.swing.boxlayout;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class DemoBoxLayoutTwo extends JFrame {
    JButton jButtonOne, jButtonTwo, jButtonThree, jButtonFour, jButtonFive;
    public DemoBoxLayoutTwo() {
        Container con=this.getContentPane();
        con.setLayout(new BoxLayout(con, BoxLayout.X_AXIS));
        jButtonOne=new JButton("按钮 1");
        jButtonTwo=new JButton("按钮 2");
```

```
jButtonThree=new JButton("特殊按钮 3");
jButtonThree.setAlignmentX(Component.RIGHT_ALIGNMENT);
jButtonFour=new JButton("按钮 4");
jButtonFive=new JButton("更大的按钮 5");
con.add(jButtonOne);
con.add(Box.createGlue()); //添加 Glue 控件
con.add(jButtonTwo);
con.add(Box.createHorizontalStrut(40)); //添加 Strut 控件
con.add(jButtonThree);
con.add(Box.createRigidArea(new Dimension(20,30)));
//添加 rigid 控件
con.add(jButtonFour);
con.add(jButtonFive);
this.setTitle("BoxLayout 示例");
this.setBounds(100,100,500,200);
this.setVisible(true);
this.validate();
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String args[]) {
    DemoBoxLayoutTwo dlayout=new DemoBoxLayoutTwo();
}
}
```

编译并运行程序，界面如图 7-26 所示。



图 7-26 程序运行界面

“按钮 1”与“按钮 2”之间插入了水平方向的 Glue 控件，增大窗口大小，如图 7-27 所示。



图 7-27 增大窗口大小

从图中可以看到，插入 Glue 控件之后，“按钮 1”与“按钮 2”之间的距离会自动增大。

“按钮 2”与“特殊按钮 3”之间插入了 Strut 控件，缩小窗口的大小，如图 7-28 所示。

可以看到缩小窗口大小以后，“按钮 2”与“特殊按钮 3”之间的距离始终保持不变。“特殊按钮 3”与“按钮 4”之间插入了 rigid 控件，调整该窗口大小以后，“按钮 3”与“按钮 4”之间的距离也保持不变。

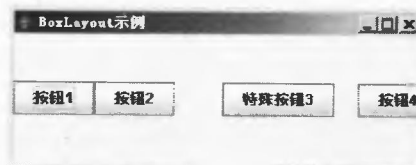


图 7-28 缩小窗口大小

7.7.3 Box 容器的嵌套使用

前面简单介绍了 Box 类的使用，从中可以看到，虽然其可以通过在可视控件之间插入一些不可视的控件来设置布局，但是其明显的缺点是所有控件只能存在于同一行或者在同一列中摆放。对于比较复杂的界面布局，显然不能仅仅通过插入非可视控件来完成。

Box 类提供了 createHorizontalBox() 和 createVerticalBox() 两个非常重要的方法。可以使用这两个方法创建在水平方向摆放控件或者在垂直方向摆放控件的 Box 容器，嵌套使用这些 Box 容器，可以搭建出比较复杂的界面。以下代码显示了如何嵌套使用 Box 容器：

```
Container container= getContentPane();
con.setLayout(new BoxLayout(con, BoxLayout.X_AXIS));
Box box1=Box.createVerticalBox();
Box box2=Box.createHorizontalBox();
con.add(box1);
con.add(box2);
```

上面代码首先将该窗口的布局管理器设置为 BoxLayout，接着将 Box 类的静态方法 createVerticalBox 与 createHorizontalBox 创建的 Box 对象 box1 和 box2 添加到 con 中。

可以看到，嵌套使用 Box 不是很复杂，下面给出了使用 Box 嵌套的完整示例：

```
package org.netbeans.swing.boxlayout;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class DemoBoxLayoutThree extends JFrame {
    public DemoBoxLayoutThree() {
        Container con=this.getContentPane();
        //将 con 的布局管理器设置为 BoxLayout
        con.setLayout(new BoxLayout(con, BoxLayout.X_AXIS));
        //创建水平方向的 Box 对象
        Box box1=Box.createVerticalBox();
        //创建垂直方向的 Box 对象
        Box box2=Box.createHorizontalBox();
        //添加"竖直接钮 1"~"竖直接钮 4"的 4 按钮
        for(int i=1;i<5;i++){
            JButton jButtonVertical=new JButton("竖直接钮"+i);
            box1.add(jButtonVertical);
        }
    }
}
```

```
}
//添加水平按钮 1~"水平按钮 4"的 4 按钮
for(int i=1;i<5;i++){
    JButton jButtonHorizontal=new JButton("水平按钮"+i);
    box2.add(jButtonHorizontal);
}
//向 con 中添加 box1 与 box2
con.add(box1);
con.add(box2);
//设置窗体的标题、大小、位置可见性以及关闭动作
this.setTitle("BoxLayout 示例");
this.setBounds(100,100,480,200);
this.setVisible(true);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.validate();
}
public static void main(String args[]) {
    DemoBoxLayoutThree dlayout=new DemoBoxLayoutThree();
}
}
```

编译并运行程序，界面如图 7-29 所示。

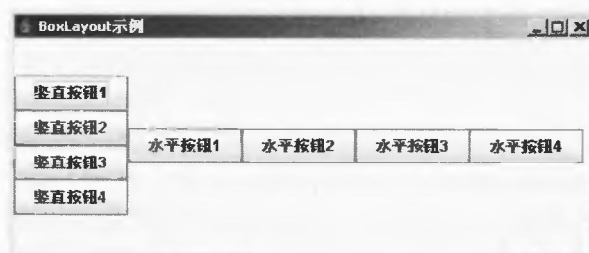


图 7-29 程序运行效果图

从图中可以看到，嵌套使用 Box 以后，添加的控件可以在水平或者垂直两个方向上摆放，而不是仅能在一个方向上摆放。

多次嵌套使用 Box，可以搭建出更复杂的界面，有兴趣的读者可以自己试验这些情况，这里就不再演示了。

7.7.4 在 NetBeans 中使用 BoxLayout 布局

在 NetBeans 中为容器设置 BoxLayout 布局非常简单。如果要为名称为 JFrameBox 的容器设置 FlowLayout 布局，首先右键单击 JFrameBox 节点，然后依次选择右键菜单的“Set Layout”/“BoxLayout”选项，即可将 JFrameBox 的布局管理器设置为 BoxLayout。

在对象观察器选中 JFrameBox 节点下面的 BoxLayout 节点，在属性窗口中可以对 BoxLayout 的 Axis 属性进行设置。

布局设置完成以后，如果向容器中添加控件，则控件会自动按照 Axis 属性值指定的顺序在容器中摆放。

7.8 使用 GroupLayout 布局管理器

使用 GroupLayout 可以很方便地进行布局管理。本节将对 GroupLayout 布局管理器进行介绍, 主要包括如下内容:

- GroupLayout 简介;
- 使用 GroupLayout 布局管理器设置间距;
- 使用 GroupLayout 布局管理器设置控件的对齐;
- 使用 GroupLayout 布局管理器进行“粘贴”。

7.8.1 GroupLayout 简介

NetBeans 中提供了一种称为 GroupLayout 的布局管理器, 该布局管理器是使用 NetBeans 开发 GUI 程序的默认布局管理器。使用 NetBeans 开发 GUI 程序, 可以通过鼠标拖曳的方式来改变控件在容器中的位置, 是一种类似于所见即所得的开发方式。将容器的布局管理器设置为 GroupLayout, 可以更好的对控件进行布局。

向使用 GroupLayout 布局管理器的容器中添加控件时, 在相应的位置会自动出现一些具有特殊功能的辅助线, 如图 7-30 所示。

这些线主要表示如下信息:

- 控件与控件或控件与边框之间的间距;
- 控件与控件之间的对齐;
- 控件与控件或控件与边框的“粘贴”情况。

如果要在 NetBeans 中为名称为 JFrameGroupLayout 的容器设置 GroupLayout 布局, 首先右键单击 JFrameGroupLayout 节点, 然后依次选择右键菜单的“Set Layout”/“Free Design”选项, 即可将 JFrameGroupLayout 的布局管理器设置为 GroupLayout。

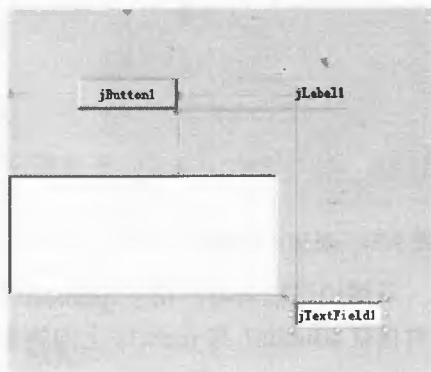


图 7-30 使用 GroupLayout 布局管理器

7.8.2 使用 GroupLayout 布局管理器设置间距

对于使用 GroupLayout 布局管理器的容器, 将控件添加到容器中时, 在控件与容器的边框之间会出现两条用于表示控件与边框之间间距的辅助线, 如图 7-31 所示。

此时, “jButton”按钮与容器的左边框和上边框之间即被指定了固定间距。单击 GUI 设计器中的“Previes Design”按钮, 界面如图 7-32 所示。调整窗口大小, 如图 7-33 所示。

控件与窗口右边框和下边框之间的间距发生了变化, 而与窗口的上边框和左边框的间距保持不变。

再向容器中添加一个控件, 并放置到容器中适当的位置, 如图 7-34 所示。

从图中可以看到, 不仅在“jLabel1”与容器的左边框之间, 增加了一条用来表示固定间距的辅助线; 而且“jButton”按钮与“jLabel1”之间, 也出现了一条辅助线, 该线表示这两个控件的间距被指定为固定值。预览该窗口, 如图 7-35 所示。改变窗口大小, 如

图 7-36 所示。

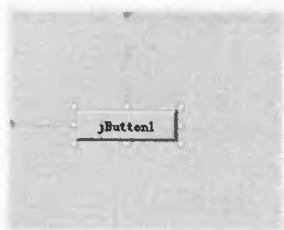


图 7-31 控件与边框间距线



图 7-32 预览窗口

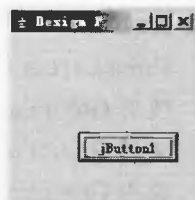


图 7-33 调整大小后窗口

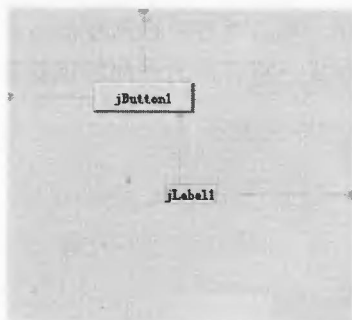


图 7-34 添加“JLabel”后窗口



图 7-35 控件与控件间距线

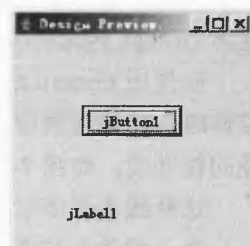


图 7-36 调整大小后窗口

从图中可以看到，由于 JLabel1 需要与窗口的右边框的距离保持不变，所以向左移动，同时保证 JButton1 与 JLabel1 之间的垂直间距始终保持不变。

7.8.3 使用 GroupLayout 布局管理器设置控件的对齐

控件与控件对齐，是通过控件中的某些线的对齐来实现的，如图 7-37 所示。

所谓的左侧对齐是指两个控件的 LEFT 线对齐，而垂直对齐是指两个控件 VCENTER 线对齐。同样，其他的对齐方式也是指控件的相应线对齐。

向图 7-35 所示窗口中继续添加控件，并放置到容器中适当的位置，如图 7-38 所示。

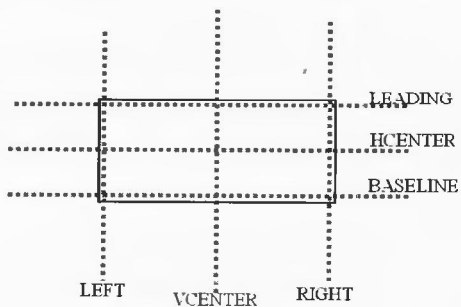


图 7-37 Swing 控件中的对齐线

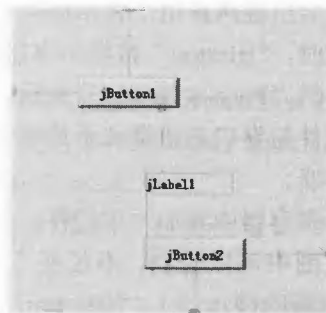


图 7-38 控件与控件对齐线

从图中可以看到,添加 `jButton2` 后,控件与边框之间不仅出现了两条用来表示控件与边框具有固定间距的辅助线,`jLabel1` 与 `jButton2` 的左边界也被一条线连接起来,该线用来表示 `jLabel1` 与 `jButton2` 的左侧永远对齐。

从图中还可以看到,添加 `jButton2` 后,`jLabel1` 不再具有图 7-33 所示的与边框保持固定间距的辅助线了,这是因为 `jLabel1` 与 `jButton2` 永远对齐,`jLabel1` 会通过 `jButton2` 与边框的间距来调整自己与边框的间距。

预览该窗口,如图 7-39 所示。调整窗口大小后,如图 7-40 所示。



图 7-39 预览窗口

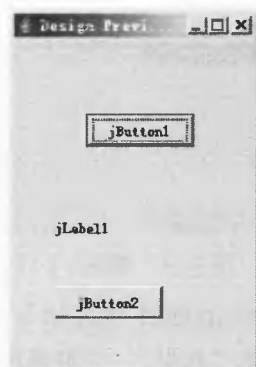


图 7-40 调整大小后窗口

比较图 7-39 与图 7-40, `jLabel1` 与 `jButton2` 的左侧始终对齐,并且 `jLabel1` 与右边框也始终保持固定的间距。

不仅可以使控件的左侧始终对齐,通过调整控件的位置,还可以使控件的右侧,控件的底部始终保持对齐,这里不再给出详细的说明,读者可以试验这几种情况。

7.8.4 使用 GroupLayout 布局管理器进行“粘贴”

对于使用 `GroupLayout` 布局管理器的容器,不仅可以设置间距与对齐,还可将控件与控件,控件与容器边框进行“粘贴”。向容器中添加控件时,当控件与边框足够近时,会出现如图 7-41 所示的辅助线。

该线即用来表示如果在当前位置放置控件,则控件会与容器的右边框进行粘贴。单击鼠标左键向容器中添加控件,如图 7-42 所示。

此时,“`jButton1`”即与容器的右边框进行了粘贴。预览该窗口,如图 7-43 所示。



图 7-41 粘贴线示意图



图 7-42 添加控件后示意图

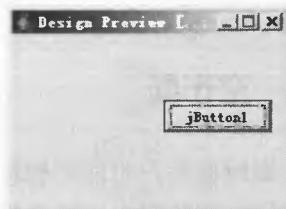


图 7-43 预览窗口

调整窗口大小,如图 7-44 所示。比较图 7-43 与图 7-44 可以看出, `jButton1` 与容器的右边框始终“粘贴”到一起。

在实际开发 GUI 程序的过程中,开发人员有时候会希望某个控件能够随着窗口大小的改变而改变,如 `JTextField`。而希望另外一些控件不会随着窗口大小的改变而改变,如 `JButton`。使用 `GroupLayout` 布局管理器提供的“粘贴”功能,可以很方便地满足这种需求。

向图 7-44 所示的窗口中继续添加一个标签 `JLabel1`,使其与窗口的左侧“粘贴”,然后在 `JLabel1` 与 `jButton1` 之间添加一个文本框 `jTextField1`,调整其大小和位置,如图 7-45 所示。

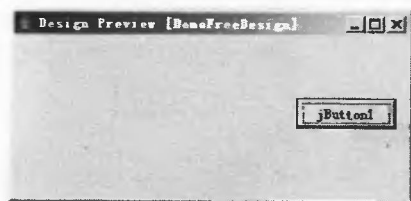


图 7-44 调整大小后窗口

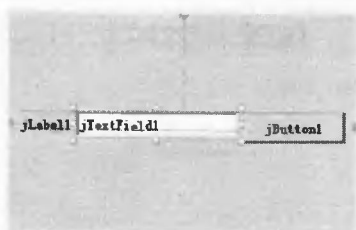


图 7-45 控件与控件粘贴线

此时, `jTextField1` 的左侧与 `jLabel1` “粘贴”,而右侧与 `jButton1` 进行“粘贴”, `jButton1` 与窗口的右侧“粘贴”。预览该窗口,如图 7-46 所示。

调整该窗口大小后,如图 7-47 所示。

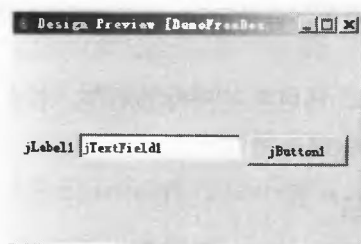


图 7-46 预览窗口

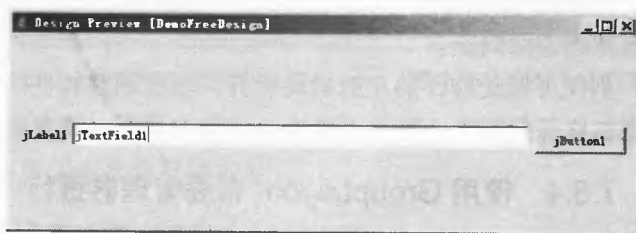


图 7-47 调整大小后窗口

从图中可以看到,调整窗口大小以后,改变大小的只是 `jTextField1`,而 `jLabel` 与 `jButton` 并没有发生改变。

对于使用 `GroupLayout` 布局管理器管理的控件来说,有些是可以改变大小的,比如上面用到的 `jTextField1`,而另外一些不应该改变大小,如 `JButton` 与 `JLabel`。不用担心,布局管理器会安排好一切。

7.9 空布局

有时候用户可能不想使用任何布局管理器,而是把控件放到容器中的固定位置上。Java 中把这种不使用布局管理器的布局称为 `null` 布局。本节将对 `null` 布局展开讨论,主要包括如下内容:

- `null` 布局简介;
- 在 NetBeans 中设置 `null` 布局。

7.9.1 null 布局简介

虽然开发 Java 程序的时候可以不使用布局管理器，但是如果可能的话，还是应该使用布局管理器。使用布局管理器将使得控件的外观更加容易，反之将有碍于程序的平台独立性。

如果一个容器所包含控件的外观感觉不会因为容器的大小、字体、外观感觉以及语言环境的变化而变化，则可以使用 null 布局对控件进行绝对定位。使用 null 布局对控件进行绝对定位的步骤如下。

- (1) 把容器的布局管理器设置为 null。
- (2) 把控件添加到容器中。
- (3) 根据需要指定控件的位置和大小。

下面代码显示了如何为容器设置 null 布局并向其中添加控件：

```
this.setLayout(null);
JLabel nullLabel=new JLabel("使用 null 布局");
nullLabel.setBounds(30,30,60,20); //设置控件的大小和位置
this.add(nullLabel);
```



提示

所谓 null 布局实际上就是没有布局管理器，所有控件的大小和位置由开发人员调用对应控件的 `setBounds`、`setSize` 和 `setLocation` 方法来确定。

7.9.2 在 NetBeans 中使用 null 布局

在 NetBeans 中为容器设置 null 布局管理器非常简单。本节将通过一个例子介绍在 NetBeans 中使用 null 布局。打开前面创建的项目 `layoutmanager`，按照如下步骤完成该例。

(1) 向 `layoutmanager` 中添加一个通过 `JFrame Form` 模版创建的名称为 `DemoNullLayout` 的类。

(2) 在对象观察器窗口中选中 `JFrame` 节点，单击鼠标右键，在弹出的菜单中依次选择“Set Layout”/“Null Layout”选项，即可将该 `JFrame` 的布局管理器设置为 null。此时再向容器中添加控件，则控件使用绝对位置放置，各个控件的大小和位置如图 7-47 所示。

(3) 向 `JFrame` 中添加 2 个 `JButton`，2 个 `JLabel`，1 个 `JTextField` 和 1 个 `JPasswordField`，并按照如表 7-8 所示修改属性。

表 7-8

属性与值对照表

控件类型	控件名称	text 属性值
JButton	jButtonEnter	确定
JButton	jButtonCancel	取消
JLabel	jLabelUserName	用户名:
JLabel	jLabelPsw	密码:
JTextField	jTextFieldUserName	空
JPasswordField	jTextFieldPsw	空

(4) 将下列代码添加到 `DemoLayout` 类的 `public static void main(String args[])` 方法中：

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoNullLayout ().setVisible(true);  
    }  
});
```

(5) 编译并运行项目，界面如图 7-48 所示。

(6) 调整窗口大小，如图 7-49 所示。

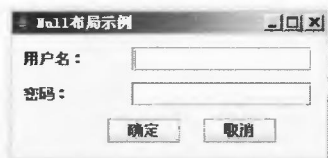


图 7-48 程序运行界面



图 7-49 调整大小后窗口

从图中可以看到，对于使用 null 布局进行布局管理的容器，调整大小后外观发生了很大的变化。因此，在目前的 GUI 程序设计中，已经很少用到 null 布局。

提示 如果一定要使用 null 布局，可以将容器的大小设置为不可更改。

7.10 用 NetBeans 开发使用复合布局管理器的程序

在前面的几节中，已经介绍了 Java 中的各种布局管理器，下面将通过一个例子，展示如何开发一个综合使用各种布局管理器的项目。

7.10.1 项目概述

在本节中将使用 NetBeans 开发一个名片管理程序，其中使用了 CardLayout、GridLayout、GroupLayout 以及 null 等几种布局管理，程序界面如图 7-50 所示。

此程序有查看状态和添加状态两种状态。当选中“查看已有名片”单选按钮时，程序处于查看状态。在查看状态中可以在“名片列表”中选择要查看的名片，程序在右边的面板中会显示该名片所包含的详细信息，在此状态中名片信息处于不可编辑状态。

当选中“添加新名片”单选按钮时，程序处于添加状态。此时所有关于名片信息的项目都处于可编辑状态，可以填写名片的基本信息，并通过选择“爱好”或者“学历”来设置不同的附加信息，设置完成以后单击“添加”按钮保存信息。此时会在“名片列表”中出现添加名片的名称。单击“清空”按钮，可以清空还没提交的内容。

在查看状态和添加状态下，都可以通过单击“爱好”或者“学历”单选按钮以显示名片的不同附加信息。



图 7-50 程序运行界面

7.10.2 界面设计

启动 NetBeans，按照如下步骤完成该例。


(1) 新建一个名称为 `cardmanage` 的项目，并将其主类名称设置为 `org.netbeans.swing.cardmanage.CardManage`。

(2) 向 `cardmanage` 项目中添加一个通过 `JFrame Form` 模板创建的 `CardManageFrame`，并将其 `title` 属性的值设置为 `CardManager Frame`。

(3) 向窗体中添加一个 `JsplitPane`，设置 `JSplitPane` 的 `divider Location` 属性的值为 150，`dividerSize` 属性的值为 3，并将 `JSplitPane` 的名称修改为 `jSplitPaneGlobal`。

(4) 分别向 `jSplitPaneGlobal` 中“左键”和“右键”所标识部分添加一个新的 `JsplitPane`，将其名称分别修改为 `jSplitPaneLeft`、`jSplit-PaneRight`。

(5) 在属性窗口中修改属性，分别将其 `orientation` 属性的值设置为 `VETICAL_SPLIT`，将 `divider Location` 属性的值设置为 110、280，并将 `dividerSize` 属性分别设置为 0、3。

(6) 向 `jSplitPaneLeft` 的“左键”所标识部分添加一个 `JPanel`。将 `JPanel` 的名称修改为 `jPanelLeftTop`。然后在“Properties”对话框中，单击“border”属性右侧的  按钮，出现如图 7-51 所示的对话框，此时即可完成 `jPanel-LeftTop` 的 `border` 属性的设置。

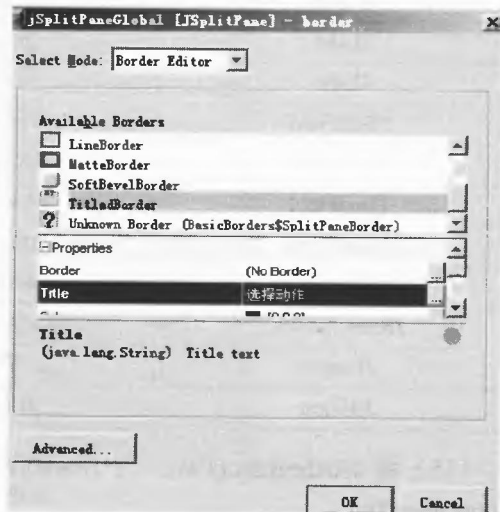


图 7-51 设置边框

(7) 在 Available Borders 列表框中选中“Title Border”选项，并将 Title 属性的值设置为“选择动作”。单击“OK”按钮完成。

Border 属性设置的是容器的边框。Swing 系列控件的边框被归入一个独立程序包 javax.swing.border 中。从 JComponent 继承的 Swing 组件都可以设置边框。反过来，

提示 不以 JComponent 为父类的顶层容器，如 JFrame 和 JDialog，就无法设置边框。组件一旦被设置边框，就会从被设置时起给自己绘制上边框，其默认的镶边也会被设置的镶边代替。通过使用不同的边框可以使程序界面更美观。

(8) 向 jPanelLeftTop 中添加 ButtonGroup，将其名称修改为 buttonGroupOne。

(9) 向 jPanelLeftTop 中添加两个 JRadioButton，将其名称分别修改为 jRadioButtonOldCard、jRadioButtonAddNewCard；将这两个单选按钮 buttonGroup 属性的值均设置为 buttonGroupOne。最后将 text 属性值分别修改为“查看已有名片”和“添加新名片”。

(10) 向 jSplitPaneLeft 的“右键”所标识的部分添加 JPanel。将该 JPanel 的名称修改为 jPanelLeftBottom。设置其 border 属性的值为 Title Border，Title 属性值为“名片列表”。

(11) 将 jSplitPaneLeft 的布局管理器设置为 GridLayout。并向 jPanelLeftBottom 添加 JList，将该 JList 的名称修改为 ListCardList，并将其 model 属性的初始值设置为空。

(12) 向 jSplitPaneRight 的“左键”所标识部分添加 JPanel。将该 JPanel 的名称修改为 jPanelRightTop。将其 border 属性值设置为 Title Border，Title 属性的值设置为“名片详细信息”。

(13) 向 jPanelLeftTop 中添加 ButtonGroup，将其名称修改为 buttonGroupTwo。

(14) 向 jPanelRightTop 中添加 4 个 JLabel、4 个 JTextField 和 2 个 JRadioButton，并按照如表 7-9 所示修改属性。

表 7-9

属性与值对照表

控件类型	控件名称	text 属性值
JLabel	jLabelName	姓名
JLabel	jLabelAddress	地址
JLabel	jLabelPhone	联系电话
JLabel	jLabelEmail	电子邮件
JTextField	jTextFieldName	空
JTextField	jTextFieldAddress	空
JTextField	jTextFieldPhone	空
JTextField	jTextFieldEmail	空
JRadioButton	JRadioButtonFavor	爱好
JRadioButton	JRadioButtonDegree	学位
JButton	JButtonAdd	添加
JButton	JButtonDelete	清空

(15) 将 JRadioButtonFavor 与 JRadioButtonDegree 单选按钮的 buttonGroup 属性均设置为 buttonGroupTwo。

(16) 向 jSplitPaneRight 的“右键”所标识的部分添加 JPanel。将该 JPanel 的名字修改为

jPanelRightBottom。设置其 border 属性为 Title Border, Title 属性的值为“名片附加信息”, 设置其布局管理器为 CardLayout。

(17) 依次向 jPanelRightBottom 中添加 2 个 JPanel, 将这 2 个 JPanel 的名称分别修改为 jPanelFavor 和 jPanelDegree, 并将 Card Name 属性值分别设置为 favorcard 与 degreecard。

(18) 向 jPanelFavor 添加 6 个 JCheckBox, 并按照如表 7-10 所示修改属性。

表 7-10 JCheckBox 属性及值对照表

控件类型	控件名称	text 属性值
JCheckBox	jCheckBoxSing	唱歌
JCheckBox	jCheckBoxDance	跳舞
JCheckBox	jCheckBoxChat	聊天
JCheckBox	jCheckBoxFootBall	足球
JCheckBox	jCheckBoxBasketBall	篮球
JCheckBox	jCheckBoxVolleyBall	排球

(19) 向 jPanelDegree 中添加 ButtonGroup, 将其名称修改为 buttonGroupThree。再向 jPanelDegree 中添加 4 个 JRadioButton, 并按照如表 7-11 所示修改属性。

表 7-11 JRadioButton 属性及值对照表

控件类型	控件名称	text 属性值
JRadioButton	jRadioButtonBachelor	学士
JRadioButton	jRadioButtonMaster	硕士
JRadioButton	jRadioButtonDoctor,	博士
JRadioButton	jRadioButtonOther	其他

(20) 分别将上面添加的 4 个单选按钮的 buttonGroup 属性设置为 buttonGroupThree。此时界面如图 7-52 所示。

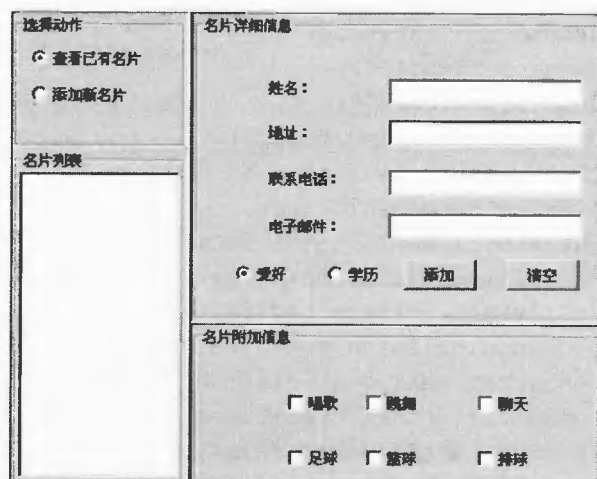


图 7-52 程序界面


完成了以上步骤, 程序界面的设计就基本完成了, 下面还要为程序编写完成特定功能的

代码。

7.10.3 功能代码的开发

(1) 向 CardManageFrame 中添加一个名称为 `setState`，返回值为 `void`，访问限制修饰符为 `private`，入口参数为 `boolean flag` 的方法，代码设置如下：

```
this.jListCardList.setEnabled(flag);
this.jTextFieldAddress.setEditable(!flag);
this.jTextFieldPhone.setEditable(!flag);
this.jTextFieldName.setEditable(!flag);
this.jButtonAdd.setEnabled(!flag);
this.jButtonDelete.setEnabled(!flag);
this.jCheckBoxDance.setEnabled(!flag);
this.jCheckBoxChat.setEnabled(!flag);
this.jCheckBoxSing.setEnabled(!flag);
this.jCheckBoxBasketBall.setEnabled(!flag);
this.jCheckBoxFootBall.setEnabled(!flag);
this.jCheckBoxVolleyBall.setEnabled(!flag);
this.jRadioButtonBachelor.setEnabled(!flag);
this.jRadioButtonDoctor.setEnabled(!flag);
this.jRadioButtonMaster.setEnabled(!flag);
this.jRadioButtonOther.setEnabled(!flag);
```

 **说明** 该方法用来设置名片信息的可编辑性。将下列代码添加到方法中。

(2) 向 CardManageFrame 中添加一个名称为 `clearAll`，返回值为 `void`，访问限制修饰符为 `private`，入口参数为空的方法，代码设置如下：

```
this.jTextFieldAddress.setText("");
this.jTextFieldPhone.setText("");
this.jTextFieldName.setText("");
this.jTextFieldEmail.setText("");
this.jCheckBoxDance.setSelected(false);
this.jCheckBoxChat.setSelected(false);
this.jCheckBoxSing.setSelected(false);
this.jCheckBoxBasketBall.setSelected(false);
this.jCheckBoxFootBall.setSelected(false);
this.jCheckBoxVolleyBall.setSelected(false);
this.jRadioButtonBachelor.setSelected(false);
this.jRadioButtonDoctor.setSelected(false);
this.jRadioButtonMaster.setSelected(false);
this.jRadioButtonOther.setSelected(false);
```

 **提示** 该方法用来清空填写但未提交的信息。并将下列代码添加到方法中。

(3) 分别为 `jRadioButtonAddNewCard`、`jRadioButtonOldCard`、`jRadioButtonFavor`、`jRadioButtonDegree` 添加 `ItemEvent` 事件处理方法，并按如下所述向相应的方法中添加代码。

- 将下列代码添加到 `jRadioButtonOldCardItemStateChanged` 方法中：

```
this.setState(false); //设置名片信息显示控件为可编辑状态
this.clearAll(); //清空名片信息显示控件中的内容
```

- 将下列代码添加到 `jRadioButtonAddNewCardItemStateChanged` 方法中：

```
this.setState(true); //设置名片信息显示控件为不可编辑状态
this.clearAll();
```

- 将下列代码添加到 `jRadioButtonFavorItemStateChanged` 方法中：

```
((java.awt.CardLayout)jPanelRightBottom.getLayout()).show(jPanelRightBottom, "favorcard");
```

- 将下列代码添加到 `jRadioButtonDegreeItemStateChanged` 方法中：

```
((java.awt.CardLayout)jPanelRightBottom.getLayout()).show(jPanelRightBottom, "degreecard");
```

(4) 向 `org.netbeans.swing.cardmanage` 包添加一个类 `CardInfo`，此类用来记录一张名片的信息。向 `CardInfo` 添加属性，并按照如表 7-12 所示修改属性值。

表 7-12 属性与值对照表

属性名称	类型	属性名称	类型
Name	String	Address	String
Phone	String	Email	String
danceState	boolean	chatState	boolean
footBallState	boolean	basketBallState	boolean
VolleyBallState	boolean	bachelorState	boolean
masterState	boolean	doctorState	boolean
otherState	boolean	singState	boolean

(5) 修改添加的成员变量的属性。Name、Address、Phone、E-mail 为 String 类型，其他都为 boolean 型，所有属性的访问限制修饰符号都设置为 `private` 的。并为所有属性添加相应的访问器。

(6) 向 `CardManageFrame` 类中添加名称为 `getCardInfo`，访问限制符为 `private`，返回值为 `CardInfo` 的方法。该方法将用户所填写的名片信息内容包装成为一个 `CardInfo` 对象返回。将以下代码添加到该方法中：

```
CardInfo cardinfo=new CardInfo();
cardinfo.setName(this.jTextFieldName.getText());
cardinfo.setAddress(this.jTextFieldAddress.getText());
cardinfo.setphone(this.jTextFieldPhone.getText());
cardinfo.setEmail(this.jTextFieldEmail.getText());
cardinfo.setSingState(this.jCheckBoxSing.isSelected());
cardinfo.setDanceState(this.jCheckBoxDance.isSelected());
```

```

cardinfo.setChatState(this.jCheckBoxChat.isSelected());
cardinfo.setFootBallState(this.jCheckBoxFootBall.isSelected());
cardinfo.setBasketBallState(this.jCheckBoxBasketBall.isSelected());
cardinfo.setVolleyBallState(this.jCheckBoxVolleyBall.isSelected());
cardinfo.setBachelorState(this.jRadioButtonBachelor.isSelected());
cardinfo.setMasterState(this.jRadioButtonMaster.isSelected());
cardinfo.setDoctorState(this.jRadioButtonDoctor.isSelected());
cardinfo.setOtherState(this.jRadioButtonOther.isSelected());
return cardinfo;

```

(7) 向 CardManageFrame 类中添加名称为 setCardInfo, 访问限制符为 private, 返回值为 void, 入口参数名称为 cardinfo 的方法。该方法用于获取 CardInfo 中的信息, 并在对应的控件中显示。将以下代码添加到该方法中:

```

this.jTextFieldName.setText(cardinfo.getName());
this.jTextFieldPhone.setText(cardinfo.getPhone());
this.jTextFieldAddress.setText(cardinfo.getAddress());
this.jTextFieldEmail.setText(cardinfo.getEmail());
this.jCheckBoxSing.setSelected(cardinfo.getSingState());
this.jCheckBoxDance.setSelected(cardinfo.getDanceState());
this.jCheckBoxChat.setSelected(cardinfo.getSingState());
this.jCheckBoxFootBall.setSelected(cardinfo.getFootBallState());
this.jCheckBoxBasketBall.setSelected(cardinfo.getBasketBallState());
this.jCheckBoxVolleyBall.setSelected(cardinfo.getVolleyBallState());
this.jRadioButtonBachelor.setSelected(cardinfo.getBachelorState());
this.jRadioButtonMaster.setSelected(cardinfo.getMasterState());
this.jRadioButtonDoctor.setSelected(cardinfo.getDoctorState());
this.jRadioButtonOther.setSelected(cardinfo.getOtherState());

```

(8) 向 CardManageFrame 类添加 vecListCard 属性, 设置属性的访问限制符为 private, 类型为 java.util.Vector, 此属性用来存储名片的姓名列表, 作为 jListCardList 列表控件的数据模型。

(9) 向 CardManageFrame 类添加 hashCardInfo 属性, 设置属性的访问限制符为 private, 类型为 java.util.Hashtable, 此属性用来存储所有已经添加的名片信息。

(10) 分别为 jButtonAdd、jButtonDelete、jListCardList 添加相应的事件处理方法, 并按如下所述向相应的方法中添加代码。

● 向 jButtonAddActionPerformed 方法的方法中添加如下代码:

```

/*
 * 该方法将用户所填写的信息包装到 CardInfo 对象中, 用“姓名”字段的值作为键名
 * 将 CardInfo 对象存储到 Hashtable 中, 同时将“姓名”字段的值存储到 Vector 中
 * 并将该 Vector 对象传给 jListCardList 以显示更新后名片的名称列表
 */
CardInfo cardinfo=this.getCardInfo();
hashCardInfo.put(cardinfo.getName(),cardinfo);
vecListCard.add(cardinfo.getName());

```

```
this.jListCardList.setListData(vecListCard);  
this.clearAll();
```

● 向 `jButtonDeleteActionPerformed` 方法的方法中添加如下代码:

```
this.clearAll(); //清空名片信息显示控件中的内容
```

● 向 `jListCardListValueChanged` 方法的方法中添加如下代码:

```
/*  
 *该方法将用户所选中名片的详细信息在控件中显示  
 */  
String selectedCardName=(String)jListCardList.getSelectedValue();  
CardInfo cardinfo=(CardInfo)hashCardInfo.get(selectedCardName);  
this.setCardInfo(cardinfo);
```

(11) 将下列代码添加到 `CardManager` 类的 `public static void main(String args[])` 方法中:

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new CardManager Frame().setVisible(true);  
    }  
});
```

(12) 编译主项目并运行, 运行界面与图 7-49 一致。

7.10.4 项目总结

在本项目中, 介绍了如何将 `JSplitPane` 控件、 `GroupLayout` 布局、 `CardLayout` 布局组合起来搭建流行的三叉戟风格的界面, 从本项目中可以看到用 `NetBeans` 集成开发环境开发具有复杂界面的程序是多么简单, 都不用编写实际界面的创建代码, 只要用鼠标进行一些简单的操作即可完成。

7.11 小结

本章结合具体的实例, 介绍了 `Swing` 中常用的几种布局管理器, 包括流布局、边框布局、箱式布局空布局等。并介绍了另外一种在 `NetBeans` 中经常使用的布局管理器 `GroupLayout` 。在本章的最后开发了一个综合使用各种布局管理器的程序, 从而帮助读者综合使用各种布局管理器。

第 8 章

Swing 对话框

与 JFrame 一样,对话框也是编写 GUI 程序时经常用到的顶层容器,很多时候都会使用对话框从用户处收集信息。本章将会对 Java 中提供的对话框以及如何在 NetBeans 中使用这些对话框进行讨论,主要包括如下内容:

- JDialog 简介;
- 使用 JOptionPane;
- 文件选择器;
- 颜色选择器;
- 在 NetBeans 中开发使用复合对话框的程序。

8.1 Swing 对话框类——JDialog

JDialog 是 Swing 包中提供容器类,可以使用其创建对话框。本节将对 JDialog 展开讨论,主要包括如下内容:

- JDialog 类简介;
- 在 NetBeans 中使用 JDialog。

8.1.1 JDialog 类简介

JDialog 类扩展了 java.awt.Dialog 类。与 Dialog 一样, JDialog 主要也是用来创建对话框的。下面使用 JDialog 创建一个最简单的对话框,代码如下:

```
JDialog myDialog=new JDialog();
```

同普通的对话框一样, JDialog 也区分模式对话框和非模式对话框。非模式的对话框允许用户同时在对话框和程序的主窗口中输入信息,而模式对话框在处理完之前不能与程序的主窗口进行交互。创建一个模式对话框的代码如下:

```
Frame myFrame=new Frame();
```

```
JDIALOG modalDialog=new JDIALOG(myFrame,true);
```

如果直接使用 JDIALOG，可以使用 add 方法向其中添加控件，使用 setVisible 方法设置可见性，使用 setLayout 方法设置布局管理等。下列代码显示了创建一个 JDIALOG 并向其中添加控件：

```
JDIALOG comJDIALOG=new JDIALOG();
JLabel myJLabel=new JLabel("向 JDIALOG 中添加 JLabel");
comJDIALOG.add(myJLabel);
```

JDIALOG 类提供了 11 个构造器，表 8-1 中给出了其中一些常用的及其说明。

表 8-1 JDIALOG 常用构造器及说明

构造器	说 明
JDIALOG(Frame owner)	创建一个没有标题但将指定的 Frame 作为其所有者的无模式对话框，如果 owner 为 null，则将一个共享的、隐藏的窗体设置为该对话框的所有者
JDIALOG(Frame owner,boolean modal)	创建一个没有标题但有指定所有者 Frame 的有模式或无模式对话框
JDIALOG(Frame owner,String title)	创建一个具有指定标题和指定所有者窗体的无模式对话框
JDIALOG(Frame owner,String title,boolean modal)	创建一个具有指定标题和指定所有者 Frame 的有模式或无模式对话框
JDIALOG(Dialog owner)	创建一个没有标题但将指定的 Dialog 作为其所有者的无模式对话框
JDIALOG(Dialog owner,String title,boolean modal)	创建一个具有指定标题和指定所有者对话框的有模式或无模式对话框

一般情况下，每一个 JDIALOG 都依赖于一个父窗口。当父窗口被销毁时，该对话框也将销毁；当父窗口被图标化时，对话框也会从屏幕中消失；当父窗口在屏幕上显示时，则对话框也会在屏幕上显示。

JDIALOG 类可以通过许多方法进行相应的操作，表 8-2 给出了其中一些常用的方法及其说明。

表 8-2 JDIALOG 类的常用方法及说明

方 法	说 明
isModal()	获取当前对话框是否是模式对话框
setModal(boolean b)	指定当前对话框是否是模式对话框
getTitle()	获得对话框的标题，标题显示在对话框的上边界上
setTitle(String title)	设置对话框的标题

另外，JDIALOG 还继承了 java.awt.Component、java.awt.Container、java.awt.Window 类的众多方法，需要用到这些方法的用户可以查阅相关的 API。

8.1.2 在 NetBeans 中使用 JDIALOG

NetBeans 对 JDIALOG 提供了全面的支持，在 NetBeans 中使用 JDIALOG 非常便捷，本节将

通过一个例子介绍如何在 NetBeans 中使用 JDialog。创建一个名称为 dialog 的项目，并将其主类名称设置为 org.netbeans.swing.dialog.Demo，然后按照如下步骤完成该例。

(1) 在项目窗口中用鼠标右键单击 org.netbeans.swing.dialog 节点，依次选择右键菜单中的“New”/“JDialog Form”选项。此时，界面如图 8-1 所示。

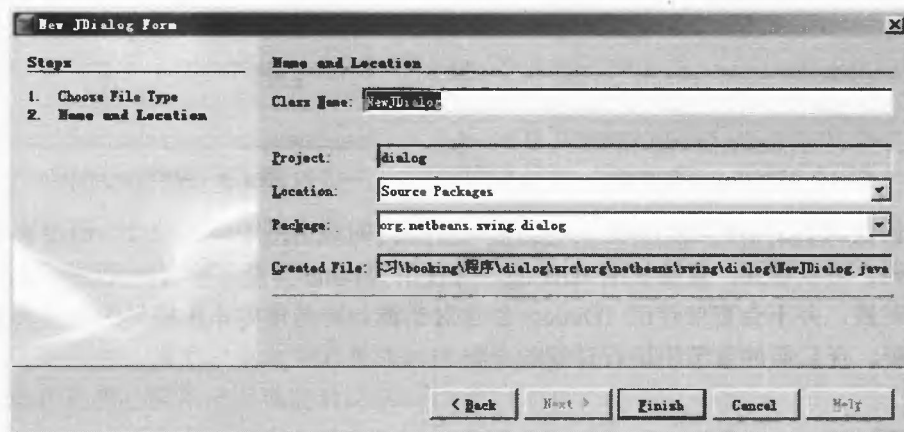


图 8-1 添加类窗口

(2) 将 Class Name 的值修改为 DemoJDialog，此值即为新添加对话框的类名。单击“OK”按钮完成添加。

(3) 此时，在 GUI 设计器中会出现一个没有任何内容的面板，该面板类似于使用“JFrame Form”模版创建类以后 GUI 设计器中出现的面板。可以向该面板中添加控件。

(4) 在对象观察器窗口中右键单击 JDialog 节点，选择右键菜单的 Properties 选项，在弹出的属性对话框中修改该 JDialog 的属性。将 alwaysOnTop 属性设置为选中状态，将 title 属性的值修改为“JDialog 应用示例”。最后，将 modal 属性的状态设置为选中，该属性用来标识该对话框是模式对话框还是非模式对话框。

(5) 向对话框中添加控件。在 Palette 窗口中选中要添加的控件，在对话框中的适当位置单击鼠标左键，即可向对话框中添加控件。在此向对话框中添加 2 个 JButton，1 个 JLabel。

(6) 按照如表 8-3 所示修改相应控件的属性。

表 8-3

属性与值对照表

控件类型	控件名称	text 属性值
JButton	jButtonCancel	取消
JButton	jButtonEnter	确定
JLabel	jLabelMessage	这是一个对话框

(7) 此时，GUI 设计器界面如图 8-2 所示。

(8) 将下列代码添加到 Demo 类中的 public static void main(String[] args)方法中：

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoJDialog ().setVisible(true);  
    }  
});
```


(9) 编译并运行主项目，界面如图 8-3 所示。



图 8-2 JDIALOG 示例界面

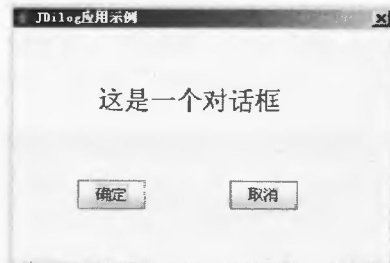


图 8-3 程序运行界面

本例仅仅演示了如何创建自定义对话框以及如何向对话框中添加控件，而没有编写具体的业务代码。可以看到，直接使用 JDIALOG，与使用 JFrame 并没有什么太大的区别。

很多时候，并不会直接使用 JDIALOG 创建对话框，而是使用系统提供的几种具有特殊功能的对话框。在后面的章节将进行详细的介绍。

8.2 使用 JOptionPane

JOptionPane 扩展了 javax.swing.JComponent，是 Swing 提供的用于收集用户信息的简单对话框。本节将对 JOptionPane 展开讨论，主要包括如下内容：

- JOptionPane 类简介；
- JOptionPane 的 4 种对话框。

8.2.1 JOptionPane 类简介

使用 JOptionPane，可以定义并创建不同类型的对话框。JOptionPane 提供了对使用标准对话框、图标、指定对话框标题和文本以及自定义按钮文本的支持。

JOptionPane 是一个用来建立复杂 JDialog 的实用工具类，其提供了一个建立普通弹出式模式对话框的简单方法。使用这种方法大大地减轻了开发人员编写代码的数量，但是也有明显的缺点，即无论用户正在做什么，都会强迫用户停止正在进行的操作并对弹出式对话框做出响应。以下代码显示了如何创建 JOptionPane：

```
String message=new String("用户名或密码错误，请重新输入");
int messageType=JOptionPane. ERROR_MESSAGE;
int optionType= JOptionPane. DEFAULT_OPTION;
JOptionPane my JOptionPane=new JOptionPane (message,messageType,optionType);
```

以上代码使用指定的内容、消息类型以及选项创建 JOptionPane。

JOptionPane 提供了 7 个构造器，表 8-4 给出了常用的 5 个及其说明。

表 8-4 JOptionPane 的常用构造器及其说明

构造器	说 明
JOptionPane(Object message)	创建一个显示指定消息的 JOptionPane 对象，使其使用 UI 提供的普通消息类型和默认选项

续表

构造器	说 明
<code>JOptionPane(Object message,int messageType)</code>	使用指定的内容和消息类型创建 <code>JOptionPane</code> 的对象,使拥默认选项
<code>JOptionPane(Object message,int messageType, int optionType)</code>	创建一个显示指定消息的 <code>JOptionPane</code> 的对象,其使用指定的消息类型和选项
<code>JOptionPane(Object message,int messageType, int optionType,Icon icon)</code>	创建一个显示指定消息的 <code>JOptionPane</code> 对象,其具有指定的消息类型、选项和图标
<code>JOptionPane(Object message,int messageType, int optionType,Icon icon,Object[] options)</code>	创建一个显示指定消息的 <code>JOptionPane</code> 的实例,其具有指定的消息类型、图标和选项。最初没有选择任何选项

创建 `JOptionPane` 对象时用到了 `JOptionPane` 中定义的一些属性,表 8-5~表 8-7 给出了其中常量,可将其分为 3 类。

- 消息类型: 指定正在被显示的消息。
- 选项类型: 指定应该为用户提供的选项。
- 选择选项: 指定用户选定的选项。

表 8-5 用来指定消息类型的 `JOptionPane` 常量

常 量	类 型	说 明
<code>ERROR_MESSAGE</code>	<code>int</code>	用于错误消息
<code>INFORMATION_MESSAGE</code>	<code>int</code>	用于信息消息
<code>PLAIN_MESSAGE</code>	<code>int</code>	用于任意消息
<code>QUESTION_MESSAGE</code>	<code>int</code>	用语问题对话框
<code>WARNING_MESSAGE</code>	<code>int</code>	用于警告消息

表 8-6 用来指定用户选项的 `JOptionPane` 常量

常 量	类 型	说 明
<code>DEFAULT_OPTION</code>	<code>int</code>	OK 按钮
<code>OK_CANCEL_OPTION</code>	<code>int</code>	OK 与 CANCEL 按钮
<code>YES_NO_CANCEL_OPTION</code>	<code>int</code>	YES、NO 和 CANCEL 按钮
<code>YES_NO_OPTION</code>	<code>int</code>	YES 和 NO 按钮

表 8-7 `JOptionPane` 的选择选项

常 量	类 型	说 明
<code>CANCEL_OPTION</code>	<code>int</code>	按下 Cancel 按钮
<code>CLOSED_OPTION</code>	<code>int</code>	按下 NO 按钮
<code>NO_OPTION</code>	<code>int</code>	按下 NO 按钮
<code>OK_OPTION</code>	<code>int</code>	按下 OK 按钮
<code>YES_OPTION</code>	<code>int</code>	按下 YES 按钮
<code>UNINITIALIZED_VALUE</code>	<code>Object</code>	指出没有为对话框设置值的值

`JOptionPane` 中提供了相应的方法对各种属性进行操作,表 8-8 给出了其中常用的方法及其说明。

表 8-8 JOptionPane 的常用方法及说明

方 法	说 明
setMessage(Object newMessage)	设置对话框中显示的消息
getMessage()	获取对话框中显示的消息
setIcon(Icon newIcon)	设置要显示的图标。如果非 null，则外观不提供图标
getIcon()	获取此对话框中显示的图标
setValue(Object newValue)	设置用户所选值
getValue()	获取用户所选值。UNINITIALIZED_VALUE 表示用户尚未作出选择，而 null 则表示用户没有选取任何项便关闭了窗口。否则，返回值将为在此对象中所定义的选项之一
setOptions(Object[] newOptions)	设置此对话框中显示的选项。如果 newOptions 中的元素为 Component，则直接将其添加到对话框中；否则创建一个按钮
getOptions()	获取用户可以做出的选择
setInitialValue(Object newInitialValue)	设置要启用的初始值，即最初显示对话框时处于焦点状态的 Component
getInitialValue()	获取初始值
setMessageType(int newType)	设置选项对话框中的消息类型
getMessageType()	获取消息类型
setOptionType(int newType)	设置要显示的选项
getOptionType()	获取显示的选项类型
setSelectionValues(Object[] newValues)	设置对话框中的输入选择值，该对话框向用户提供可以从中进行选择的项列表。null 值表示用户可以输入想输入的任何值，通常依靠 JTextField 来实现
getSelectionValues()	获取输入选择值
setInitialSelectionValue(Object newValue)	设置（根据选择）最初向用户显示的输入值
getInitialSelectionValue()	获取（根据最初选择）向用户显示的输入值
setInputValue(Object newValue)	设置由用户选择或输入的输入值
getInputValue()	获取用户已输入的值

另外，JOptionPane 类还提供了一些用来创建特殊对话框的静态方法（对象工厂），在下面的小节中，将会重点讲述这些方法的使用。

8.2.2 JOptionPane 的 4 种对话框

使用 JOptionPane 中提供的 showXXXDialog 静态方法，可以用来建立 4 种不同的对话框。

- MessageDialog：向用户显示消息，其只包含一个按钮。
- ConfirmDialog：向用户显示任意数据，其可能包含用户可以从中进行选择的按钮。
- OptionDialog：要求用户确认某些信息，其包括 YES、NO、OK、Cancel 这样的按钮。
- InputDialog：为用户提供了输入数据的一些方法，其包含“OK”与“Cancel”两个按钮。

图 8-4~图 8-7 给出了这几种对话框的典型界面。

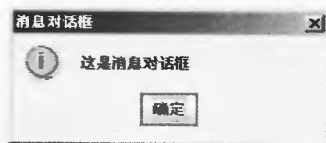


图 8-4 消息对话框

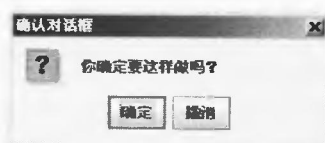


图 8-5 确认对话框

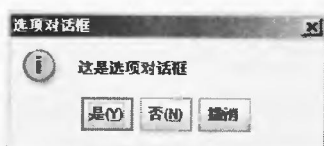


图 8-6 选项对话框

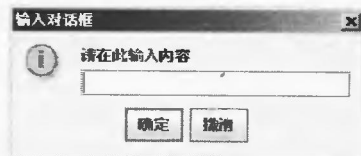


图 8-7 输入对话框

为了创建 4 种不同的对话框, JOptionPane 提供了 4 种 showXXXDialog 方法, 并为创建每种对话框都提供了多种重载方法, 这些方法的参数遵循一致的模式如下。

- **parentComponent**: 指定作为此对话框的父对话框的容器。
- **message**: 设置要在该对话框中显示的描述消息。通常情况下, message 就是一个 String 字符串。不过, 此参数的类型实际上是 Object。
- **messageType**: 定义 message 的类型。表 8-5 给出了 messageType 的值。
- **optionType**: 定义在对话框中显示的选项按钮的集合, 表 8-6 给出了 optionType 的值。
- **options**: 对将在对话框中显示的选项按钮集合的更详细描述。options 参数经常使用的是 String 数组, 但是参数类型是 Object 数组。
- **icon**: 要在对话框中显示的图标。图标的默认值由 messageType 参数确定。
- **title**: 对话框的标题。
- **initialValue**: 默认选择 (输入值)。

下面给出了创建这 4 种对话框最简单的方法的签名。

- 创建确认对话框: `showConfirmDialog(Component parentComponent, Object message)`。
- 创建输入对话框: `showInputDialog(Component parentComponent, Object message)`。
- 创建消息对话框: `showMessageDialog(Component parentComponent, Object message)`。
- 创建选项对话框: `showOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)`。

对于每种方法的重载方法, 其参数的含义已经在前面给出了, 需要用到这些方法的读者, 请读者查阅有关的 API, 这里就不再给出了。创建一个确认对话框, 其代码如下:

```
JFrame fatherFrame=new JFrame();
String message=new String("你确定要这样做吗? ");
String title=new String("确认对话框");
JOptionPane.showConfirmDialog(fatherFrame,message,title,JOptionPane.
OK_CANCEL_OPTI
ON, JOptionPane.QUESTION_MESSAGE,null);
```

下面给出了使用输入对话框的完整代码:

```
package org.netbeans.swing.dialog;  
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
public class DemoInputDialog extends JFrame{  
    JLabel jLabelMessage;  
    public DemoInputDialog() {  
        //创建一个 JLabel,设置其对齐方式与字体,并添加到主窗口中  
        jLabelMessage=new JLabel();  
        jLabelMessage.setHorizontalAlignment(SwingConstants.CENTER);  
        jLabelMessage.setFont(new Font("宋体",Font.PLAIN,28));  
        this.add(jLabelMessage);  
        //设置主窗口的大小、位置、可见性、标题  
        this.setBounds(150,150,400,200);  
        this.setTitle("输入对话框应用示例");  
        this.setVisible(true);  
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);  
        //使用给定的标题、消息类型、消息内容创建 InputDialog  
        String title=new String("输入对话框");  
        String message=new String("请在下面输入内容");  
        int messageType=JOptionPane.INFORMATION_MESSAGE;  
        String inputMessage=(String)JOptionPane.showInputDialog(this,  
            message,title,messageType);  
        //获取在 InputDialog 中输入的文字并在 jLabelMessage 中显示  
        if(inputMessage!=null && !inputMessage.equals("")){  
            this.jLabelMessage.setText("你输入了: "+inputMessage);  
        }else{  
            this.jLabelMessage.setText("你没有输入任何内容!");  
        }  
    }  
    public static void main(String args[]){  
        new DemoInputDialog();  
    }  
}
```

编译并运行程序,如图 8-8 所示。在其中输入“这是一个输入对话框”后,窗口如图 8-9 所示。

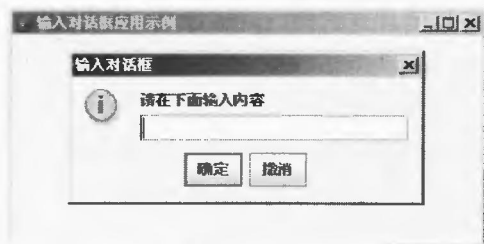


图 8-8 输入对话框

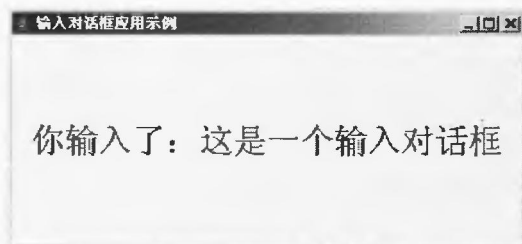


图 8-9 显示输入内容

对于其他几种对话框，使用方法与 `InputDialog` 基本相同。用户可以参照该例试验其他几种情况，这里就不再给出具体的示例了。

8.3 文件选择器

当编写应用程序的时候，经常需要打开和保存文件。一个好的文件选择器是很难编写的，其应该能够显示文件和目录并且能够让用户浏览文件系统，对文件类型进行过滤等。

Swing 提供了一个 `JFileChooser` 类，其提供了一个 GUI，用来浏览文件系统并提供通过一个列表选择文件或目录，或者输入文件和目录的名称选择文件和目录的功能。

`JFileChooser` 并不是 `JDialog` 的子类，其直接扩展了 `javax.swing.JComponent`。`JFileChooser` 通过调用 `showOpenDialog` 方法来显示一个用于打开文件的对话框，调用 `showSaveDialog` 方法来显示一个用于保存文件的对话框，调用 `showDialog` 方法打开具有自定义按钮的对话框。

图 8-10 显示了一个文件选择器界面。

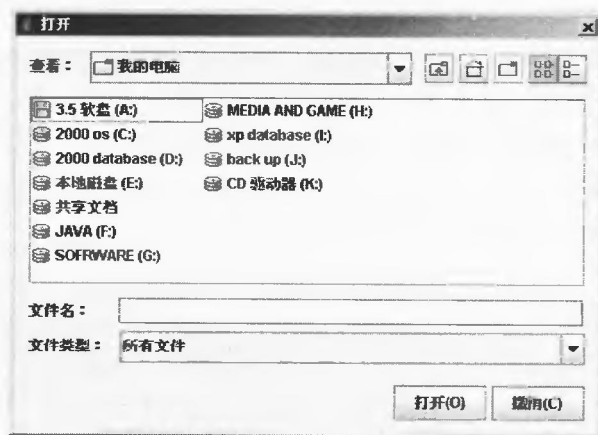


图 8-10 文件选择器

以下代码显示了如何创建 `JFileChooser` 对象，并打开一个用于保存文件的对话框：


```
JFileChooser myJFileChooser=new JFileChooser();
myJFileChooser.showSaveDialog(father); //father 为该对话框的父控件
```

`JFileChooser` 类一共提供了 6 个构造器，表 8-9 给出了其中常用的 3 个及其说明。

表 8-9 `JFileChooser` 的常用构造器及其说明

构造器	说明
<code>JFileChooser()</code>	构造一个指向用户默认目录的 <code>JFileChooser</code> 。此默认目录取决于操作系统。在 Windows 上通常是“我的文档”，在 Unix 上是用户的主目录
<code>JFileChooser(String currentDirectoryPath)</code>	构造一个使用给定路径的 <code>JFileChooser</code>
<code>JFileChooser(File currentDirectory)</code>	使用给定的 <code>File</code> 作为路径来构造一个 <code>JFileChooser</code>

从表 8-9 可以看到，创建 `JFileChooser` 对象与创建 `JDialog` 对象不同。前者不用在构造器中指定父控件。这允许在多个窗口中复用同一个文件选择器。

 **提示** JFileChooser 的构造器可能运行得很慢，尤其是在 Windows 上用户映射了很多网络路径的情况下，此时使用复用文件选择器就会节省部分时间。

JFileChooser 类提供了很多方法进行相应的操作，表 8-10 给出了常用的方法及其说明。

表 8-10 JFileChooser 的常用方法及说明

构造器	说 明
getSelectedFile()	获取选中的文件。由程序员通过 setFile 设置，或者通过用户选择来确定返回的内容
setSelectedFile(File file)	设置选中的文件。如果该文件的父目录不是当前目录，则将当前目录更改为该文件的父目录
getSelectedFiles()	如果将文件选择器设置为允许选择多个文件，则返回选中文件的列表
setSelectedFiles(File[] selectedFiles)	如果将文件选择器设置为允许选择多个文件，则设置选中文件的列表
getCurrentDirectory()	获得当前目录
setCurrentDirectory(File dir)	设置当前目录。传入 null 会设置文件选择器指向用户的默认目录
showOpenDialog(Component parent)	弹出一个打开文件选择器
showSaveDialog(Component parent)	弹出一个保存文件选择器
showDialog(Component parent, String approveButtonText)	弹出具有自定义标签按钮的自定义文件选择器对话框
getDialogType()	获得此对话框的类型，默认值是 JFileChooser.OPEN_DIALOG
setDialogType(int dialogType)	设置此对话框的类型，参数值可以为 JFileChooser.OPEN_DIALOG, JFileChooser.SAVE_DIALOG, JFileChooser.CUSTOM_DIALOG
setDialogTitle(String dialogTitle)	设置显示在 JFileChooser 窗口标题栏的字符串
getDialogTitle()	获得 JFileChooser 的标题栏中所显示的字符串
setFileSelectionMode(int mode)	设置 JFileChooser，以允许用户只选择文件、只选择目录，或者可选择文件和目录。默认值是 JFileChooser.FILES_ONLY, JFileChooser.DIRECTORIES_ONLY, JFileChooser.FILES_AND_DIRECTORIES
getFileSelectionMode()	获得当前的文件选择模式
isFileSelectionEnabled()	根据当前的文件选择模式确定文件是否为可选择的
isDirectorySelectionEnabled()	根据当前的文件选择模式确定目录是否为可选择的
setMultiSelectionEnabled()	设置文件选择器，以允许选择多个文件
isMultiSelectionEnabled()	如果可以选择多个文件，则返回 true
setFileFilter(FileFilter filter)	设置当前文件过滤器。文件选择器使用文件过滤器从用户的视图中过滤文件
getFileFilter()	获得当前选择的文件过滤器
getName(File f)	获得文件名
getDescription(File f)	获得文件描述
accept(File f)	如果应该显示该文件，则返回 true

Java 提供了一个叫做 javax.swing.filechooser.FileFilter 的类，通过扩展该类，可以很容易

地对文件选择器中显示的文件类型进行过滤，代码设置如下：

```
//创建一个自定义的文件过滤器类
class PngFliter extends FileFilter{
    //该方法用来测试文件是否被接受
    public boolean accept(File file){
        //如果要打开的是png格式的文件，或者是一个目录，则该文件被接受
        return file.getName().toLowerCase().endsWith(".png") || file.isDirectory();
    }
    //该方法返回文件类型的说明信息
    public String getDescription(){
        return "Png Image";
    }
}
//向 JFileChooser 对象添加文件类型过滤器
fileChooser.addChoosableFileFilter (new PngFliter());
```

可以看到，对文件选择器中显示的文件类型进行过滤，只需要提供一个扩展了 `FileFilter` 的类，并在该类中重写 `accept` 与 `getDescription` 两个方法，然后再使用 `addChoosableFileFilter` 方法将该类的对象添加到文件选择器列表中即可。

8.4 颜色选择器

`JColorChooser` 也是 GUI 程序设计中经常用到的一种控件，使用其可以迅速地开发出颜色选择器对话框，本节将对 `JFileChooser` 进行讨论，主要包括如下内容：

- `JColorChooser` 类简介；
- 在 NetBeans 中开发使用 `JFileChooser` 的程序。

8.4.1 JColorChooser 类简介

除了文件选择器以外，Swing 还提供了一种颜色选择器——`JColorChooser`，可以使用颜色选择器来挑选颜色。颜色选择器既可以作为单独的对话框使用，同时也可以作为控件放置在任何自定义的界面当中。使用颜色选择器作为控件，使得开发一个包含颜色选择器的界面非常简单。典型的颜色选择器如图 8-11 所示。

默认情况下，颜色选择器提供了 3 个选择器面板。

- 样品：用来从样品颜色集合中选择颜色；
- HSB：使用色彩—饱和度—亮度颜色模型来选择颜色；
- RGB：使用红—绿—蓝颜色模型来选择颜色。

同 `JFileChooser` 类一样，`JColorChooser` 直接扩展了 `java.swing.JComponent`，而没有扩展 `JDialog`。下面代码显示了如何创建一个初始颜色为白色的颜色选择器：

```
JColorChooser jcc=new JColorChooser(Color.white);
```

`JColorChoose` 类有 3 个构造器，如表 8-11 所示。

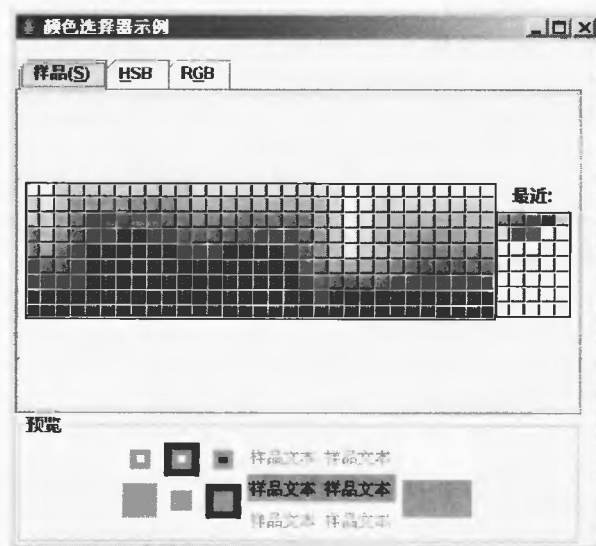


图 8-11 典型文件选择器

表 8-11

JColorChooser 构造器及说明

构造器	说 明
JColorChooser()	创建初始颜色为白色的颜色选取器
JColorChooser(Color initialColor)	创建具有指定初始颜色的颜色选取器
JColorChooser(ColorSelectionModel model)	创建具有指定 ColorSelectionModel 颜色选取器

JColorChoose 类提供了很多方法进行相应的操作,表 8-12 给出了其中常用的方法及说明。

表 8-12

JColorChoose 类的常用方法及说明

方 法	说 明
getColor()	获取颜色选取器的当前颜色值
setColor(Color color)	将颜色选取器的当前颜色设置为参数指定的颜色
setColor(int r,int g,int b)	将颜色选取器的当前颜色设置为指定的 RGB 颜色。红色、绿色和蓝色的值应该介于数字 0~255 之间 (其中包括 0 和 255)
setColor(int c)	在选取器中设置当前颜色的整数值,其中 8 个低位指定 Blue 值,接下来的 8 个位指定 Green 值,再往上的 8 个位指定 Red 值
setPreviewPanel(JComponent preview)	设置当前预览面板。此操作将激发 PropertyChangeEvent
getPreviewPanel()	返回显示选取颜色的预览面板
getSelectionModel()	返回处理颜色选择的数据模型
setSelectionModel(ColorSelectionModel newModel)	设置包含所选颜色的模型

颜色选择器使用 ColorSelectionModel 对象来包含并管理当前的选择。当用户改变颜色选择器中的颜色时, ColorSelectionModel 将会产生 ChangeEvent 事件。下面代码是为 ColorSelectionModel 添加监听器, 并处理 ChangeEvent 事件:

```
colorChooser.getSelectionModel().addChangeListener(this);
.....
public void stateChanged(ChangeEvent e) { //业务代码}
```

提示 当事件源的状态发生改变的时候,就会产生 `javax.swing.event.ChangeEvent` 事件,使用 `ChangeListener` 接口中的 `stateChanged` 方法来处理该事件。

8.4.2 在 NetBeans 中开发 JColorChooser 的程序

在前面的章节中对 `JColorChooser` 进行了简单的介绍。在本小节中,将通过一个例子介绍如何在 NetBeans 中开发使用 `JColorChooser` 的程序。打开前面创建的项目 `dialog`,按照如下步骤完成该例。

(1) 向 `dialog` 中添加一个通过 `JFrame Form` 模版创建的名称为 `DemoJColorChooser` 的类。

(2) 将 `DemoJColorChooser` 的 `title` 属性的值修改为“颜色选择器示例程序”。并将 `DemoJColorChooser` 的布局管理器设置为 `BorderLayout`。

(3) 向 `DemoJColorChooser` 的 `North` 区域中添加一个 `JPanel`,将其名称修改为 `jPanelNorth`,将 `jPanelNorth` 的 `border` 属性的值修改为 `TitledBorder`,并将 `border` 的 `title` 属性的值修改为“显示颜色变化”。

(4) 向 `JPanel` 中添加一个 `JLabel`,将其名称修改为 `jLabelMessage`,并将其 `text` 属性的值修改为“欢迎使用颜色选择器”。

(5) 向 `DemoJColorChooser` 的 `CENTER` 区域中添加一个 `JColorChooser`,将其名称修改为 `jColorChooserCenter`。将 `jColorChooserCenter` 的 `border` 属性的值修改为 `TitledBorder`,并将 `border` 的 `title` 属性的值修改为“颜色选择器”,此时 GUI 设计器界面如图 8-12 所示。

(6) 向 `DemoJColorChooser.java` 文件中添加一个名称为 `ChooseAction` 的类,在“Add New Class”对话框中将“final”复选框设置为非选中状态。单击“Add”按钮,弹出如图 8-13 所示的窗口。

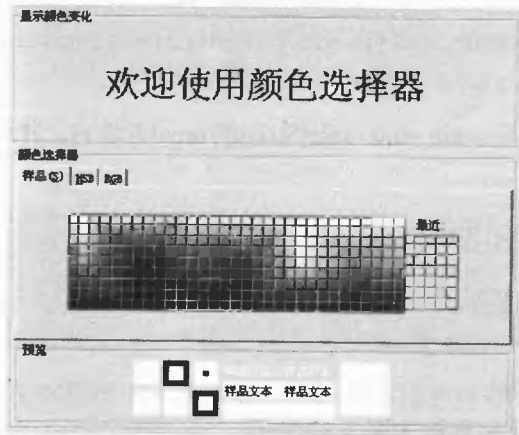


图 8-12 程序设计界面

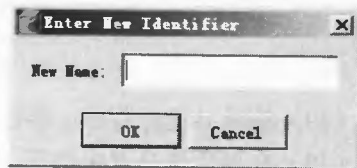


图 8-13 添加类实现的接口

(7) 在如图 8-13 所示的窗口中输入“`javax.swing.event.ChangeListener`”,单击“OK”按

钮。在 Object 列表框中出现如图 8-14 所示的内容。

(8) 此时窗口中的内容即为 ChooseAction 类实现的接口。单击“Add New Class”对话框中的“OK”按钮，完成添加。此时，源代码编辑器窗口中出现如图 8-15 所示的内容。

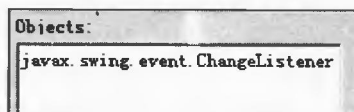


图 8-14 已经添加的接口

```
class ChooseAction implements javax.swing.event.ChangeListener {  
}
```

图 8-15 源代码编辑器中内容

(9) 向 ChooseAction 类的类体内添加如下代码，以使用颜色选择器的当前颜色设置控件的前景色：

```
JComponent component;  
JColorChooser colorChooser;  
public ChooseAction(JComponent component, JColorChooser colorChooser) {  
    this.component=component; //获得传入的 JComponent 参数  
    this. colorChooser=colorChooser; // 获得传入的 JColorChooser 参数  
}  
//实现 ChangeListener 接口中的方法  
public void stateChanged(ChangeEvent e) {  
    Color newColor=this.colorChooser.getColor(); //获得 JColorChooser 的当前颜色  
    component.setForeground(newColor); //设置 JComponent 的前景色  
}
```

(10) 向 DemoJColorChooser 类的构造器内添加如下代码：

```
//创建监听器对象  
ChooseAction caListenr=new ChooseAction(jLabelMessage,jColorChooser  
Center);  
//为颜色选择器的模型添加监听器  
this.jColorChooserCenter.getSelectionModel().addChangeListener(caLis-  
tenr);
```

(11) 将下列代码添加到 Demo 类中的 public static void main(String[] args)方法内，并删除原来方法中的代码：

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoJColorChooser ().setVisible(true);  
    }  
})
```

(12) 编译并运行程序，界面如图 8-16 所示。

(13) 在颜色选择器中选择一定的颜色，此时界面如图 8-17 所示。

还可以通过在 HSB 与 RGB 面板中选择颜色来改变 jLabelMessage 中字体的颜色，这里不再赘述。



图 8-16 程序运行界面



图 8-17 在颜色选择器中选择一定的颜色

8.5 实战：在 NetBeans 中开发复合对话框的程序

前面几节介绍了 Java 中的一些对话框，本节中将通过一个例子介绍如何在 NetBeans 中开发一个综合使用对话框的例子。

8.5.1 项目概述

在本节中将使用 NetBeans 开发一个文本阅读器程序，程序界面如图 8-18 所示。

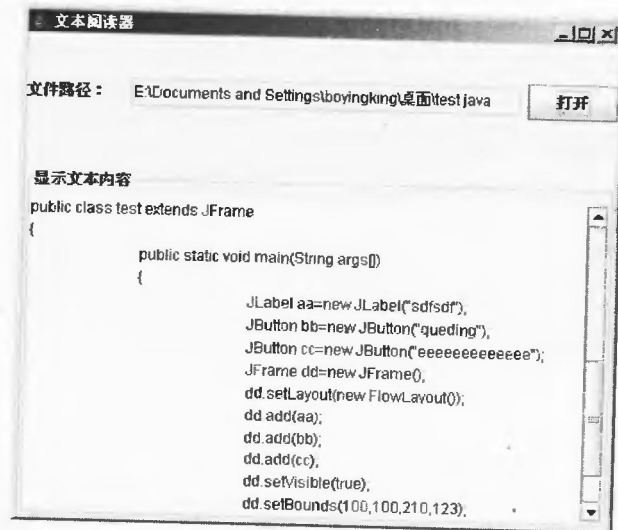


图 8-18 程序界面

在本程序中，可以通过“打开”按钮打开文件选择器，在文件选择器中选择要打开的文件。在操作的过程中，针对用户的不同操作会出现一些与用户交互的对话框，比如询问用户是否确认打开，给用户提示一些信息等。

本程序只能打开 Java 文件或者 txt 文本文件，对于其他类型能的文件，并不在文件选择

器的文件列表中显示。窗口上方的文本框用来显示当前打开文件的路径，不可编辑。窗口下方的文本区用来显示当前打开文件的内容，也是不可编辑的。

8.5.2 界面设计

打开本节中前面创建的项目 `dialog`，按照如下步骤完成该例。

(1) 向 `dialog` 中添加一个通过 `JFrame Form` 模版创建的名称为 `DemoReader` 的类。

(2) 将 `DemoReader` 的 `title` 属性的值修改为“文本阅读器”。并将 `DemoReader` 的布局管理器设置为 `BorderLayout`。

(3) 向 `DemoReader` 的 `North` 区域添加一个 `JPanel`，将其名称修改为 `jPanelNorth`。

(4) 向 `jPanelNorth` 中各添加一个 `JLabel`、`JButton` 和 `JTextField`，并按照表 8-13 修改相应的属性。

表 8-13

属性与值对照表

控 件 类 型	控 件 名 称	text 属性值
<code>JLabel</code>	<code>jLabelFilePath</code>	文件路径
<code>JButton</code>	<code>jButtonOpen</code>	打开
<code> JTextField</code>	<code> jTextFieldFilePath</code>	空

(5) 修改 `jLabelFilePath` 的 `editable` 属性，将其设置为不可编辑。

(6) 向 `DemoReader` 的 `CENTER` 区域中添加一个 `JTextArea`，将其名称修改为 `jTextAreaContent`。并将其 `lineWrap` 与 `wrapStyleWord` 属性都设置为选中状态。并修改 `editable` 属性的值，使其不可编辑。

(7) 在对象观察器窗口中选中 `jScrollPane1` 节点，该节点表示将 `JTextArea` 添加到 `DemoReader` 时自动添加的 `JScrollPane`，将该 `JScrollPane` 的名称修改为 `jScrollPaneCenter`，将其 `border` 属性的值修改为 `TitledBorder`，并将 `TitledBorder` 的 `Title` 属性的值修改为“显示文本内容”。

(8) 此时 GUI 设计器中界面如图 8-19 所示。

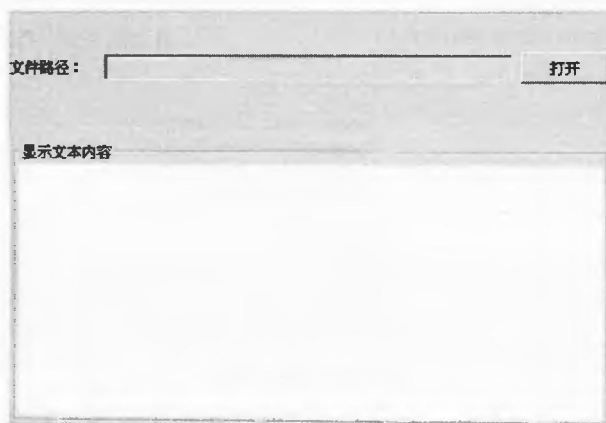


图 8-19 程序设计界面

完成了以上步骤，程序界面的设计就基本完成了。下面就要为程序编写完成特定功能的代码。

8.5.3 功能代码的开发

(1) 向 DemoReader.java 文件中添加一个扩展了 javax.swing.filechooser.FileFilter 的类, 将该类的名称设置为 TxtFliter, 并将下列代码添加到该类中:

```
boolean flag;  
//设置文件选择器的列表中只显示 txt 文件或者目录  
public boolean accept(File file){  
    if(file.getName().toLowerCase().endsWith(".txt")) {  
        flag=true;  
    }else if(file.isDirectory()){  
        flag=true;  
    }else{  
        flag=false;  
    }  
    return flag;  
}  
//该方法返回文件类型的说明信息  
public String getDescription(){  
    return "TXT 文件";  
}
```

- 该类扩展了 FileFilter, 对文件选择器中显示的文件类型进行限制, 这里设置文件选择器中显示的文件类型 txt 文件或目录。

(2) 再 DemoReader.java 文件中添加一个扩展了 javax.swing.filechooser.FileFilter 的类, 将该类的名称设置为 JavaFliter。将下列代码添加到该类中:

```
boolean flag;  
//设置文件选择器的列表中只显示 Java 文件或者目录  
public boolean accept(File file){  
    if(file.getName().toLowerCase().endsWith(".java")){  
        flag=true;  
    }else if(file.isDirectory()){  
        flag=true;  
    }else{  
        flag=false;  
    }  
    return flag;  
}  
//该方法返回文件类型的说明信息  
public String getDescription(){  
    return "Java 文件";  
}
```

说明 该方法设置文件选择器中显示的文件类型为 Java 文件或目录。

(3) 向 DemoReader 中添加一个名称为 fileChooser, 类型为 javax.swing.JFileChooser, 访问限制修饰符为 private, 初始值为 new javax.swing.JFileChooser() 的成员变量。

(4) 向 DemoReader 中添加签名为 public void initFileChooser() 的方法, 并将下列代码添加到该方法中:

```
//获取原来已有的过滤器
FileFilter[] fileFilter= fileChooser.getChoosableFileFilters();//获取原来已有的过滤器
int fSize=fileFilter.length;
for(int i=0;i<fSize;i++){//删除原来已有的过滤器
    fileChooser.removeChoosableFileFilter(fileFilter[i]);
}
fileChooser.addChoosableFileFilter(new TxtFliter());//添加 Java 过滤器
fileChooser.addChoosableFileFilter(new JavaFliter());//添加 Txt 过滤器
```

- 该方法获得了文件选择器中已有的过滤器并将其删除。
- 添加了自己定义的两个过滤器。

(5) 将下列代码添加到 DemoReader 的构造器内。

```
this.initFileChooser ();
```

(6) 为 jButtonOpen 添加(ActionEvent) 事件处理方法, 并将下列代码添加到事件处理方法中:

```
fileChooser.showOpenDialog(this);
File selectFile=fileChooser.getSelectedFile();
if(selectFile==null)
//用户单击了文件选择器中的取消按钮, 出现提示消息并退出
{
    JOptionPane.showMessageDialog(this,"你没有选择要打开的文件!", "提示:",
        JOptionPane.INFORMATION_MESSAGE);
    return;
}else
//出现确认对话框要求用户确认是否打开该文件
{
    int choose=JOptionPane.showConfirmDialog(this,"你确定要打开该文件吗", "请确认",
        JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
    if(choose==JOptionPane.NO_OPTION){//取消打开该文件, 出现提示对话框并退出
        JOptionPane.showMessageDialog(this,"你取消了打开文件!", "提示:",
            JOptionPane.INFORMATION_MESSAGE);
        return;
    }else
    //则读取选择的文件并在 jTextAreaContent 中显示该文件
    {
        String tempString="";//创建字符串对象, 用来存储从文件中读取的一行内容
        BufferedReader buffRead=null;//创建 BufferedReader 对象
        String pathName=selectFile.getPath();//获取文件路径
```



```

this.jTextFieldFilePath.setText(pathName);//在 jTextField
FilePath 中显示路径
try{
    int size=(int)selectFile.length();
    byte[] tempArray=new byte[size];
    FileInputStream fin=new FileInputStream(selectFile);
    fin.read(tempArray);
    String tempString=new String(tempArray);
    this.jTextAreaContent.setText(tempString);
}catch(IOException e){
    e.printStackTrace();
}
}
}
}

```

(7) 将下列代码添加到 Demo 类中的 public static void main(String[] args)方法内，并删除原来方法内的代码：

```

java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoReader ().setVisible(true);
    }
}

```

(8) 编译并运行项目，如图 8-20 所示。单击打开按钮，界面如图 8-21 所示。



图 8-20 程序运行界面



图 8-21 打开文件

说明 在图 8-20 可以看到，使用了文件过滤器以后，文件类型列表中原来的文件类型都已经不存在了。

(9) 选中“test.java”文件，单击“打开”按钮，界面如图 8-22 所示。

此时，主窗口处于不可编辑状态。图 8-23 所示为单击“否”按钮后的界面。

该程序还有其他几种运行状态，这里就不一一给出了，有兴趣的读者可以试验这些情况。

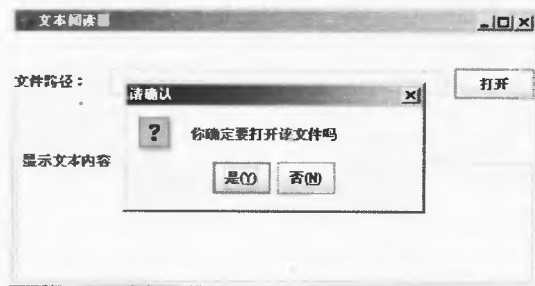


图 8-22 确认是否打开文件

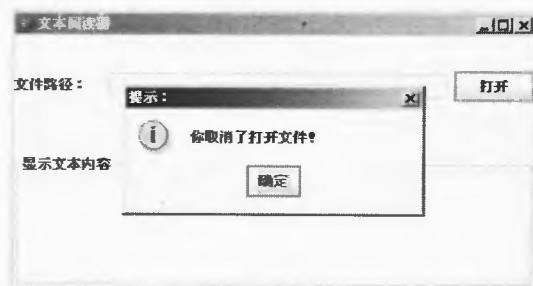


图 8-23 取消打开文件

8.5.4 项目总结

在本项目中，综合使用了文件选择器、确认对话框、消息对话框来创建应用程序。可以看到，在 NetBeans 中创建与使用对话框都是很简单的事情，不需要开发人员写很多的代码。

通过在程序中使用对话框，增加了人机交互，使得程序更加人性化。在以后开发应用程序的时候，应该在适当的地方使用对话框。

8.6 小结

本章通过具体的实例，讲述了如何使用 Swing 提供的各种对话框，包括 JOptionPane、文件选择器、颜色选择器。可以看到，在 NetBeans 中开发使用对话框的程序非常方便，只需要通过鼠标的拖曳即可完成大部分界面的开发，而不需要编写太多的代码。

第 9 章

Swing 菜单

学习使用一个新的工具，常常会从菜单学起。使用菜单可以让具有多个选项的程序界面布局更加合理。菜单是 Swing 提供的非常重要的控件，本章将对 Swing 的菜单进行介绍，主要包括如下内容：

- Swing 菜单简介；
- JMenuBar；
- JMenuItem；
- JMenu；
- 在 NetBeans 中使用菜单控件；
- JCheckBoxMenuItem 与 JRadioButtonMenuItem；
- JPopupMenu。

9.1 Swing 菜单简介

菜单是编程中经常用到的一种控件，本节将对菜单控件进行介绍，主要包括如下内容：

- Swing 菜单控件说明；
- 一个简单的菜单程序。

9.1.1 Swing 菜单控件说明

Swing 菜单控件是 JComponent 类的子类，因此其具有 Swing 控件的所有优点。下面列出了 Swing 菜单控件的一些显著特性。

- 可以在菜单中使用图标。
- 菜单项可以为单选按钮或者复选框。
- 可以为菜单项指定加速键和快捷键。
- 大多数的标准控件都可以用做菜单项。

另外，Swing 菜单为程序提供了大家熟悉的菜单分隔线、弹出式菜单和子菜单等功能。

Swing 通过调整容器的边框, 把菜单栏置于 Swing 容器的顶部, 图 9-1 所示为典型的 Swing 菜单。

位于窗口顶部的菜单栏包含了下拉菜单的名称, 单击该名称可以打开其中包含的菜单项和子菜单项的菜单。当用户单击某一菜单项的时候, 其他的菜单都会关闭。

图 9-2 是与菜单相关类的继承关系图。

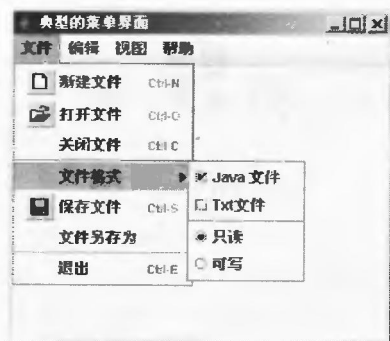


图 9-1 典型的菜单界面

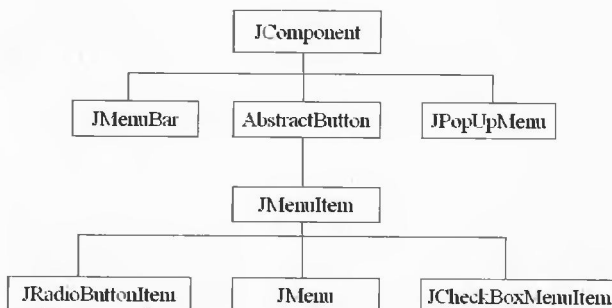


图 9-2 菜单相关类的继承关系图

下面给出了图 9-2 中菜单相关类的简单说明。

- 菜单 (JMenu) 和菜单项 (JMenuItem) 都扩展了 AbstractButton 类。菜单和菜单项包含与许多 Swing 按钮相同的特性。比如, 菜单项可以在鼠标经过时, 高亮显示; 可以被单击, 表明用户做出了选择; 可以像按钮一样被禁止使用。
- 复选菜单项 (JCheckBoxMenuItem) 与单选菜单项 (JRadioButtonMenuItem) 扩展了 JMenuItem, 其也具有与许多 Swing 按钮相同的特性, 如可以在选中与非选中状态之间进行切换。
- JMenu 扩展了 JMenuItem 类, 每个 JMenu 对象都包含一个用做菜单标题的隐藏菜单项。菜单不一定出现在菜单栏上, 其可以被嵌入其他的菜单中充当子菜单。菜单栏包含一个或多个菜单, 其使用与平台相关的位置摆放, 通常位于窗口的上方。
- 弹出式菜单 (JPopupMenu) 只有在用户执行了与平台相关的鼠标操作后才能出现。例如, 在一个支持弹出式菜单的控件上用鼠标右键单击, 在鼠标所在处将弹出快捷菜单。

对于图 9-2 中给出的每一个菜单控件, 在本章后面的小节中都会进行详细地介绍。

9.1.2 一个简单的菜单程序

上个小节中对 Swing 菜单控件进行了简单地说明, 本节中给出了一个使用菜单的简单程序, 代码如下:

```
import java.awt.*;
import javax.swing.*;
public class DemoSimpleMenu extends JFrame{
    JMenuBar jMenuBarOne;
    JMenu fileMenu, editMenu, newFileMenu;
    JMenuItem openFile, closeFile; //声明菜单
```

```
JCheckBoxMenuItem newFileTxt,newFileJava;
JRadioButtonMenuItem copyFile,pasteFile;
ButtonGroup buttonEdit;
public DemoSimpleMenu() {
    jMenuBarOne=new JMenuBar();
    buttonEdit=new ButtonGroup();
    /*
    *创建几个 JMenu 对象
    */
    fileMenu=new JMenu("文件");
    editMenu=new JMenu("编辑");
    newFileMenu =new JMenu("新建文件");
    openFile=new JMenuItem("打开文件");
    closeFile=new JMenuItem("关闭文件");
    /*
    *创建两个 JCheckBoxMenuItem 对象
    */
    newFileTxt=new JCheckBoxMenuItem("文本文件");
    newFileJava=new JCheckBoxMenuItem("Java 文件");
    /*
    *创建两个 JRadioButtonMenuItem 对象
    */
    copyFile=new JRadioButtonMenuItem("复制");
    pasteFile=new JRadioButtonMenuItem("粘贴");
    buttonEdit.add(copyFile);
    buttonEdit.add(pasteFile);
    this.setJMenuBar(jMenuBarOne);
    jMenuBarOne.add(fileMenu);
    jMenuBarOne.add(editMenu);
    /*
    *向 fileMenu 中添加 JMenuItem 对象
    */
    fileMenu.add(openFile);
    fileMenu.add(closeFile);
    fileMenu.addSeparator();
    fileMenu.add(newFileMenu);
    newFileMenu.add(newFileTxt);
    newFileMenu.add(newFileJava);
    editMenu.add(copyFile);
    editMenu.add(pasteFile);
    /*
    *设置窗体的大小、位置、可见性以及按下关闭按钮时的默认动作
    */
    this.setTitle("菜单的简单使用");
    this.setBounds(100,100,250,150);
    this.setVisible(true);
}
```

```
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
public static void main(String args[]){  
    new DemoSimpleMenu();  
}  
}
```

编译并运行程序，界面如图 9-3 所示。选择“文件”/“新建文件”选项，界面如图 9-4 所示。



图 9-3 程序运行示意图

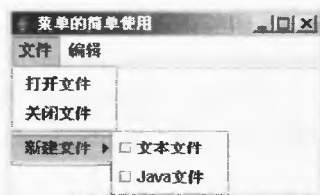


图 9-4 展开子菜单

上面的程序用到了菜单栏 (JMenuBar)、菜单项、菜单、复选菜单项以及单选菜单项。在后面的章节中将会详细地为读者讲述这些控件的使用。

9.2 菜单栏——JMenuBar

JMenuBar 用来创建一个水平的菜单栏，本节将会对 JMenuBar 进行介绍，主要包括如下内容：

- JMenuBar 简介。
- 在 NetBeans 中使用 JMenuBar。

9.2.1 JMenuBar 简介

JMenuBar 扩展了 javax.swing.JComponent 类，其功能与 java.awt.MenuBar 基本相同，都是用来创建一个水平菜单栏。开发人员可以使用 JMenuBar 类的 add 方法向菜单栏中添加菜单，JMenuBar 为添加到其中的菜单分配一个整数索引，并会根据该索引将菜单从左到右依次显示。创建菜单栏是非常简单的，代码如下：

```
JMenuBar myJMenuBar=new JMenuBar();
```

另外，还可以使用 JMenuBar 的另外一个构造器创建带有菜单的菜单栏，代码如下：

```
JMenuBar meunJMenuBar=new JMenuBar(new JMenu("Edit"));
```

创建完菜单栏以后，在通常情况下，可以使用 JDialog、JApplet 或 JFrame 类的 setJMenuBar 方法将菜单栏添加到窗体中。以下代码给出了如何将前面创建的 myJMenuBar 添加到 JDialog 中：

```
JDialog myJDialog=new JDialog();  
myJDialog.setJMenuBar(myJMenuBar);
```

提示 菜单栏是一个可以放置在窗体中的任何位置的控件,对于用上述方法添加到窗体的菜单栏,一般显示在窗体的顶部。

除了使用上述方法向窗体中添加菜单栏以外,还可以使用 `add` 方法将菜单栏添加到窗体中,下面代码显示了将 `myJMenuBar` 添加到 `JFrame` 中:

```
JFrame myJFrame=new JFrame();
myJFrame.add(myJMenuBar, BorderLayout.SOUTH);
```

此时,菜单栏被置于窗体的 NORTH 区域。

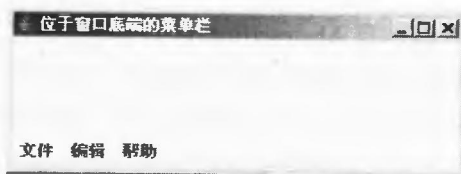


图 9-5 位于窗口底部的菜单栏

提示 开发人员至少要给菜单栏中添加一个有名称的菜单,这样菜单栏才能具备一定的高度。否则,将显示为一条与分割线相似的细线。

`JMenuBar` 类提供了很多方法进行相应的操作,表 9-1 给出了一些常用的方法及说明。

表 9-1 `JMenuBar` 的常用方法及说明

方 法	说 明
<code>add(JMenu c)</code>	将指定的菜单添加到菜单栏的末尾
<code>getMenu(int index)</code>	获取菜单栏中指定位置的菜单
<code>getMenuCount()</code>	获取菜单栏上的菜单数
<code>setHelpMenu(JMenu menu)</code>	设置用户选择菜单栏中的“帮助”选项时显示的帮助菜单
<code>getHelpMenu()</code>	获取菜单栏的帮助菜单
<code>setSelected(Component sel)</code>	设置当前选择的组件,更改选择模型
<code>isSelected()</code>	如果当前已选择了菜单栏的组件,则返回 <code>true</code>

另外 `JMenuBar` 还可以使用父类中的很多方法,这里就不再给出了,需要用到的话可以查阅相关的 API。

9.2.2 在 NetBeans 中使用 `JMenuBar`

在 NetBeans 中使用 `JMenuBar` 控件十分简单,只需要通过鼠标拖曳即可向容器中添加 `JMenuBar`。例如,要向一个 `JFrame` 中添加 `JMenuBar`,只需要在 `Palette` 窗口中选中 `JMenuBar` 选项,然后在 GUI 设计器中单击即可完成添加。

成功添加 `JMenuBar` 后,该 `JMenuBar` 会自动放置在 `JFrame` 的顶端,并且其中自动包含了一个 `text` 属性值为 `Menu` 的菜单。

在对象观察器窗口中选中 `JMenuBar`,在属性对话框中可以修改其相应的属性。对于如何向 `JMenuBar` 中添加菜单、菜单项,将会在下面的章节中进行介绍。

9.3 菜单项——JMenuItem

开发具有菜单的程序时，菜单项是不可或缺的一种控件。本节将对菜单项进行简单地介绍，主要包括如下内容：

- 菜单项简介。
- 为菜单项添加快捷键和加速器。

9.3.1 菜单项简介

JMenuItem 本质上是一个继承自 `AbstractButton` 的按钮，不过它又不完全等同于按钮。当鼠标经过某个菜单项时，Swing 就认为该菜单项被选中，但并不会触发任何事件；当用户在菜单项上释放鼠标，此时 Swing 也会认为该选项被选中，并触发事件完成相应的操作。

1. 菜单项的构造器及常用方法说明

Swing 的菜单项可以包含图像或者完全由图像构成，通过向 JMenuItem 的构造器传递一个 `Icon` 对象，即可创建一个带图标的菜单项，下面代码显示了如何创建一个带有文本 `Open` 和图标 `open.gif` 的菜单项。

```
JMenuItem iconJMenuItem=new JMenuItem("Open",new ImageIcon("open.gif"));
```

JMenuItem 一共提供了 6 个构造器，表 9-2 给出了其中常用的 4 个及其说明。

表 9-2 JMenuItem 构造器及其说明

构造器	说 明
<code>JMenuItem(Icon icon)</code>	创建带有指定图标的 JMenuItem
<code>JMenuItem(String text)</code>	创建带有指定文本的 JMenuItem
<code>JMenuItem(String text,Icon icon)</code>	创建带有指定文本和图标的 JMenuItem
<code>JMenuItem(String text,int mnemonic)</code>	创建带有指定文本和快捷键的 JMenuItem

默认情况下，新创建的 JMenuItem 的文字位于图像的左侧，开发人员可以使用 `setHorizontalTextAlignment` 方法用来改变文本与图像的位置，代码如下：

```
iconJMenuItem.setHorizontalTextAlignment(SwingConstants.RIGHT);
```

除了上面介绍的几个方法之外，JMenuItem 还提供了一些方法供开发人员使用，表 9-3 给出了这些方法及其说明。

表 9-3 JMenuItem 的常用方法及说明

方 法	说 明
<code>init(String text,Icon icon)</code>	利用指定文本和图标初始化菜单项
<code>addMenuDragMouseListener(MenuDragMouseListener l)</code>	将 <code>MenuDragMouseListener</code> 添加到菜单项，该接口中的方法用来处理 <code>MenuDragMouseEvent</code> 事件
<code>removeMenuDragMouseListener(MenuDragMouseListener l)</code>	从菜单项中移除 <code>MenuDragMouseListener</code>

续表

方 法	说 明
<code>getMenuDragMouseListeners()</code>	获取利用 <code>addMenuDragMouseListener</code> 添加到此 <code>JMenuItem</code> 的所有 <code>MenuDragMouseListener</code> 组成的数组
<code>addMenuKeyListener(MenuKeyListener l)</code>	将 <code>MenuKeyListener</code> 添加到菜单项, 该接口中的方法用来处理 <code>MenuKeyEvent</code> 事件
<code>removeMenuKeyListener(MenuKeyListener l)</code>	从菜单项中移除 <code>MenuKeyListener</code>
<code>getMenuKeyListeners()</code>	返回利用 <code>addMenuKeyListener</code> 添加到此 <code>JMenuItem</code> 的所有 <code>MenuKeyListener</code> 的数组

提示 `JMenuItem` 还可以使用父类中提供的很多方法进行相应的操作, 需要的用户可以查阅相关的 API, 这里不再赘述。

2. 菜单项事件及其处理方法

`JMenuItem` 触发的事件有很多种, 主要包括如下两种:

- `ActionEvent` 事件, 其在菜单被选中时触发;
- `ChangeEvent` 事件, `JMenuItem` 状态变化时被触发。

提示 为上述两个事件添加或删除监听器的方法都继承自 `AbstractButton` 类。

当鼠标在 `JMenuItem` 中拖动时, 会触发 `MenuDragMouseEvent` 事件。当菜单项接收到一个按键事件时, `JMenuItem` 还会产生 `MenuKeyEvent`。表 9-3 已经给出了为这两个事件注册监听器的方法。

提示 对于 `MenuDragMouseEvent` 与 `MenuKeyEvent` 的详细介绍, 需要用到的读者可以查阅其 API。

9.3.2 为菜单项添加快捷键和加速器

为菜单项添加快捷键和加速器, 用户在使用时可以提高工作效率。下面将对快捷键和加速器进行简单的介绍。

1. 快捷键

通过使用 `JMenuItem` 类的构造器 `JMenuItem(String text,int mnemonic)` 可以指定该菜单项的快捷键, 下面代码创建了一个设置了快捷键 ‘O’ 的菜单项。

```
JMenuItem mnemJMenuItem=new JMenuItem("Open File",'O');
```

快捷键会自动在菜单项中显示出来, 其方式是菜单项标签字符串中与快捷键字母相同的第一个字母加下划线, 如图 9-6 所示。

菜单项 “Open File” 的 “O” 字母有下划线, 这表明它是该菜单项的快捷键。同样, “C” 是 “Close File” 的快捷键。如果菜单项标签字符串中不包含快捷键字母, 通过该快捷键仍然



图 9-6 使用快捷键

可以选择该菜单项，但是在菜单项上却不能显示该快捷键。

说明

通过快捷键调用菜单项时要将“Alt”键和快捷键同时按下，如果快捷键为“O”则按下“Alt+O”组合键。

2. 加速器

除了在菜单项中使用快捷键外，开发人员还可以使用 `setAccelerator` 方法为菜单项指定一个加速器。加速器和快捷键的区别是：快捷键用来从当前打开的菜单中选择一个子菜单或者菜单项，而加速器可以在不打开菜单的情况下选择菜单项。

下面为将“Ctrl+O”设置为菜单项加速器的代码。

```
myJMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,  
    InputEvent.CTRL_MASK)); // myJMenuItem 为已有的菜单项
```

当加速器被添加到菜单项上时，对应的加速键就会自动地显示在相应的菜单项上，如图 9-7 所示。

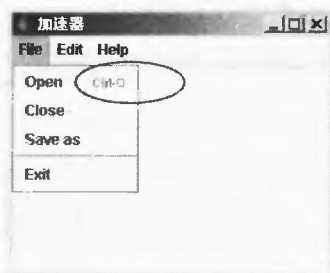


图 9-7 使用加速器

当用户按下了加速器键时，相应的菜单项就会被选择，同时触发一个动作事件，如同使用鼠标选择了该菜单项一样。事实上加速器并没有打开菜单，只是激活同菜单关联的事件。

9.4 菜单——JMenu

JMenu 是可以连接到 JMenuBar 对象或者其他 JMenu 对象上的菜单。直接添加到 JMenuBar 上的菜单叫做顶层菜单，连接到其他 JMenu 对象上的菜单称为子菜单。典型的非顶层 JMenu 都有右箭头标记，表明当用户选择该 JMenu 时，在 JMenu 旁还会弹出子菜单，如图 9-8 所示。

下面给出了一些关于 JMenu 的简单说明。

- 可以把 JMenu 看作组合控件，由 JMenuItem 与弹出式菜单（后面小节将会介绍）两种控件组合而成。JMenuItem 作为 JMenu 的标题，而单击 JMenuItem 则会在其下方放或者右侧出现弹出式菜单。



图 9-8 菜单与子菜单

- JMenu 对象给添加到其中的每一个菜单项都赋予一个整数索引，并依据索引使用菜单布局管理器调整菜单项的顺序。
- JMenu 是 JMenuItem 的子类，除了可以使用 add 方法向 JMenu 中添加菜单项或子菜单之外，还可以使用 add 方法向菜单中添加简单的字符串，此时 JMenu 对象会自动地创建具有相应标题的 JMenuItem 对象。

下面代码创建了两个 JMenu，并将其中的一个作为另一个的子菜单。

```
JMenu fatherJMenu=new JMenu("父菜单");
JMenu sonJMenu=new JMenu("子菜单");
fatherJMenu.add(sonJMenu);
```

JMenu 类提供了很多方法供开发人员使用，表 9-4 给出了其中一些常用的方法及说明。

表 9-4 JMenu 类方法及其说明

方 法	说 明
setModel(ButtonModel newModel)	设置菜单的数据模型
isSelected()	如果菜单当前是被选中的，则返回 true
setSelected(boolean b)	设置菜单的选择状态
isPopupMenuVisible()	如果菜单的弹出菜单可见，则返回 true
setPopupMenuVisible(boolean b)	设置弹出菜单的可见性。如果未启用菜单，则此方法无效
getPopupMenuOrigin()	计算 JMenu 的弹出菜单的原点。此方法使用名为 Menu.menuPopupOffsetX、Menu.menuPopupOffsetY、Menu.submenuPopupOffsetX 和 Menu.submenuPopupOffsetY 的外观属性调整弹出的确切位置
setMenuLocation(int x,int y)	设置弹出菜单的位置
add(JMenuItem menuItem)	将某个菜单项追加到此菜单的末尾，并返回添加的菜单项
add(Component c)	将组件追加到此菜单的末尾，并返回添加的控件
add(Component c,int index)	将指定控件追加到此容器的给定位置上，如果 index 等于-1，则将控件追加到末尾
add(String s)	创建具有指定文本的菜单项，并将其追加到此菜单的末尾
addSeparator()	将新分隔符追加到菜单的末尾
insert(String s,int pos)	在给定的位置插入一个具有指定文本的新菜单项
insert(JMenuItem mi,int pos)	在给定的位置插入指定的 JMenuItem
insertSeparator(int index)	在指定的位置插入分隔符
getItem(int pos)	获得指定位置的 JMenuItem，如果位于 pos 的组件不是菜单项，则返回 null
getItemCount()	获得菜单上的项数，包括分隔符
remove(JMenuItem item)	从此菜单移除指定的菜单项，如果不存在弹出菜单，则此方法无效
remove(int pos)	从此菜单移除指定索引处的菜单项
isTopLevelMenu()	如果菜单是“顶层菜单”，则返回 true
getPopupMenu()	获得与此菜单关联的弹出菜单，如果不存在将创建一个弹出菜单
addMenuListener(MenuListener l)	添加菜单事件的侦听器
getMenuListeners()	获得利用 addMenuListener()添加到此 JMenu 的所有 MenuListener 组成的数组

如果想要为 JMenu 创建快捷键，可以调用从 JMenuItem 中继承的 setMnemonic 方法。

提示 另外，JMenu 还可以使用其父类中的很多方法进行相应的操作，需要用到的用户可以查阅相关的 API。

9.5 在 NetBeans 中使用菜单控件

前面的几节对 JMenuBar、JMenuItem 和 JMenu 进行了详细地介绍。本节将通过一个例子介绍如何在 NetBeans 中使用这些控件。启动 NetBeans，按照如下步骤完成该例。

(1) 创建一个名称为 menu 的项目，并将其主类的名称设置为 org.netbeans.swing.menu.Demo。

(2) 向 menu 中添加一个通过 JFrame Form 模版创建的名称为 DemoJMenuOne 的类。

(3) 向 DemoJMenuOne 中添加一个 JMenuBar，将其名称修改为 jMenuBarGlobal。


(4) 在对象观察器窗口中选中 jMenuBarGlobal 节点下的 jMenu1 节点，该 JMenu 是添加 JMenuBar 时自动添加的，将其名称修改为 jMenuFile。在属性对话框中将其 text 属性的值修改为 File。

(5) 在对象观察器窗口中用鼠标右键单击 jMenuBarGlobal 节点，在弹出的菜单中选择“Add JMenu”选项，向 jMenuBarGlobal 中添加一个 JMenu。将该 JMenu 的名称修改为 jMenuEdit，并将其 text 属性的值修改为 Edit。

(6) 按步骤 (5) 所述再向 jMenuBarGlobal 中添加一个名称为 jMenuAbout，text 属性值为 About 的 JMenu。

(7) 此时，GUI 设计器中内容如图 9-9 所示。

(8) 在对象观察器中用鼠标右键单击 jMenuFile 节点，依次选择右键菜单中的“Add”/“JMenuItem”选项，向 jMenuFile 中添加一个 JMenuItem，将其名称修改为 jMenuItemOpenFile，将其 text 属性的值修改为 Open File，并通过修改 icon 属性的值为 jMenuItemOpenFile 添加图标。

(9) 为 jMenuItemOpenFile 设置加速器。在 jMenuItemOpenFile 的属性对话框中，单击 accelerator 属性右侧的  按钮，弹出如图 9-10 所示的窗口。

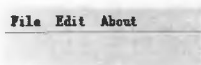


图 9-9 菜单栏

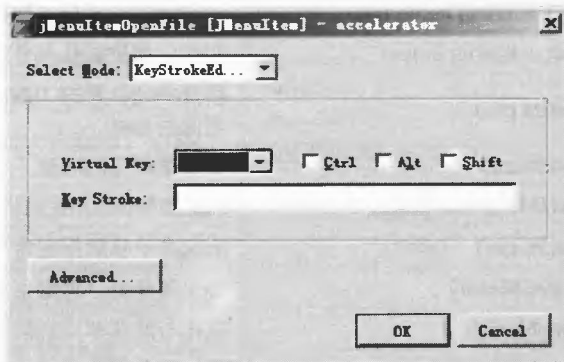


图 9-10 设置加速键窗口

(10) 通过在 Virtual Key 列表框中选择要使用的按键或者单击键盘上的相应按键修改

Key Stroke 文本框的值为 jMenuItemOpenFile 设置加速器的按钮。还可以通过选择“Ctrl”、“Alt”或者“Shift”复选框设置加速器的组合键。这里将使用“Ctrl+O”作为加速器的组合键。单击“OK”按钮完成设置。

(11) 向 jMenuFile 中添加名称为 jMenuItemSaveFile 的 JMenuItem, 将 text 属性的值修改为 Save File, 修改 icon 属性的值并为其添加图标。按照步骤(10)所述为 jMenuItemSaveFile 设置加速器, 并将加速器组合键设置为“Ctrl+S”。

(12) 在对象观察器窗口中用鼠标右键单击 jMenuFile 节点, 依次选择右键菜单中的“Add”/“JSeparator”选项, 向 jMenuFile 添加一个 JSeparator (菜单分隔条)。

(13) 向 jMenuFile 中添加名称为 jMenuItemOthers 的 JMenuItem, 将其 text 属性值修改为 Others。

(14) 向 jMenuItemOthers 中添加名称分别为 jMenuItemSetting 和 jMenuItemSaveAs 的 JMenuItem, 并将其 text 属性的值分别修改为 Setting 和 SaveAs。

(15) 向 jMenuFile 中添加名称为 jMenuItemExit 的 JMenuItem, 将 text 属性的值修改为 Exit, 并将加速器组合键设置为“Ctrl+E”。

(16) 分别为 jMenuItemOpenFile、jMenuItemSaveFile 和 jMenuItemExit 添加 ActionEvent 事件的处理方法, 并向相应方法添加如下代码。

● 向 jMenuItemOpenFileActionPerformed 方法中添加如下代码:

```
JFileChooser fileChooser=new JFileChooser();  
fileChooser.showOpenDialog(this); //打开文件选择器对话框
```

● 向 jMenuItemSaveFileActionPerformed 方法中添加如下代码:

```
JFileChooser fileChooser=new JFileChooser();  
fileChooser.showSaveDialog(this); //打开文件保存对话框
```

● 向 jMenuItemExitActionPerformed 方法中添加如下代码:

```
System.exit(0);
```

(17) 将下列代码添加到 Demo 类的 public static void main(String[] args)方法中:

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoJMenuOne().setVisible(true);  
    }  
});
```

(18) 编译并运行项目, 界面如图 9-11 所示。

(19) 依次选择“File”/“Others”选项, 界面如图 9-12 所示。

(20) 选择“File”/“Open”选项, 打开文件选择器对话框, 界面如图 9-13 所示。

提示 本程序并没有真正地从选择的文件中读取内容。

(21) 在如图 9-11 所示状态下, 按下“Ctrl+S”组合键后, 会打开文件保存对话框, 界面如图 9-14 所示。

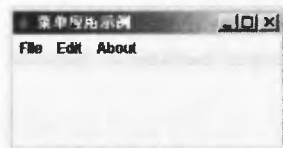


图 9-11 程序运行界面



图 9-12 选取 Others 选项



图 9-13 打开文件选择器



图 9-14 打开文件保存对话框

(22) 通过选择 File 菜单中的 Exit 选项，或者按下“Ctrl+E”组合键，即可退出程序。

9.6 单选与复选菜单项

前面几节介绍了 JMenuBar、JMenuItem 以及 JMenu 这几种菜单中经常使用的控件。除此之外，Swing 还提供了另外的两种菜单项，即 JCheckBoxMenuItem（复选菜单项）与 JRadioButtonMenuItem（单选菜单项），本节将为读者介绍这两种菜单项，主要包括如下内容：

- 复选菜单项 JCheckBoxMenuItem 简介。
- 单选菜单项 JRadioButtonMenuItem 简介。
- 在 NetBeans 中使用 JCheckBoxMenuItem 与 JRadioButtonMenuItem。

9.6.1 复选菜单项——JCheckBoxMenuItem

JCheckBoxMenuItem 是实现了复选框功能的菜单项，具有 JCheckBox 的一切特征。JCheckBoxMenuItem 可以实现同时选中几个菜单的功能。像常规菜单项一样，JCheckBoxMenuItem 可以有与之关联的文本或图标，或者二者兼而有之。

下面的代码创建了一个带有文本、图标以及给定选中状态的 JCheckBoxMenuItem：

```
ImageIcon ii=new ImageIcon("blacktitle.jpg");  
JCheckBoxMenuItem myJCheckBoxMenuItem=new JCheckBoxMenuItem("黑色",ii,true);
```


JCheckBoxMenuItem 类一共提供了 7 个构造器, 表 9-5 给出了其中常用的 5 个及其说明。

表 9-5 JCheckBoxMenuItem 的常用构造器及说明

构造器	说明
JCheckBoxMenuItem(Icon icon)	创建一个带图标的、初始未被选定的复选框菜单项
JCheckBoxMenuItem(String text)	创建一个带文本的、初始未被选定的复选框菜单项
JCheckBoxMenuItem(String text, Icon icon)	创建带有指定文本和图标的、初始未被选定的复选框菜单项
JCheckBoxMenuItem(String text, boolean b)	创建具有指定文本和选择状态的复选框菜单项
JCheckBoxMenuItem(String text, Icon icon, boolean b)	创建具有指定文本、图标和选择状态的复选框菜单项

JCheckBoxMenuItem 类本身并没有提供特别的方法, 主要使用从父类中继承的方法进行操作。其中的很多方法在前面的章节中已经进行过详细地介绍, 这里就不再重复给出了, 需要用到的用户可以查阅相关的 API。

提示 JCheckBoxMenuItem 具有很多复选框 (JCheckBox) 的功能, 具体的使用方法和复选框相同, 读者可以参照本书中介绍复选框的章节自行学习。

9.6.2 单选菜单项——JRadioButtonMenuItem

JRadioButtonMenuItem 类是实现了单选按钮功能的菜单项, 其具有 JRadioButton 类的一切特征。使用 JRadioButtonMenuItem 主要是为了实现几选一的功能, 所以其主要被编组使用, 这与 JRadioButton 相同。

下面的代码创建了一个带有文本、图标以及给定状态的 JRadioButtonMenuItem:

```
ImageIcon ii=new ImageIcon("boy.jpg");
JRadioButtonMenuItem myJRadioButtonMenuItem
    =new JRadioButtonMenuItem ("男",ii,true);
```

要使得添加到同一组中的 JRadioButtonMenuItem 具有排它性, 则需要使用 ButtonGroup 进行编组, 具体方法如下代码所列:

```
ButtonGroup buttonGroup=new ButtonGroup();
buttonGroup.add(myJRadioButtonMenuItem);
```

JRadioButtonMenuItem 类一共提供了 8 个构造器, 表 9-6 给出了其中常用的 5 个及其说明。

表 9-6 JRadioButtonMenuItem 类的常用构造器

构造器	说明
JRadioButtonMenuItem(Icon icon)	创建一个带图标的 JRadioButtonMenuItem
JRadioButtonMenuItem(String text)	创建一个带文本的 JRadioButtonMenuItem
JRadioButtonMenuItem(String text, Icon icon)	创建一个具有指定文本和图标的单选按钮菜单项
JRadioButtonMenuItem(Icon icon, boolean selected)	创建一个具有指定图像和选择状态的单选按钮菜单项
JRadioButtonMenuItem(String text, Icon icon, boolean selected)	创建一个具有指定的文本、图像和选择状态的单选按钮菜单项

与 JCheckBoxMenuItem 一样, JRadioButtonMenuItem 类本身也没有提供特别常用的方法, 主要也是使用从父类继承的方法对 JRadioButtonMenuItem 进行操作。需要用到这些方法的用户请查阅相关的 API。

提示 JRadioButtonMenuItem 具有很多单选按钮 (JRadioButton) 的功能, 具体使用的方法和单选按钮相同, 读者可以参照本书中介绍单选按钮的章节。

9.6.3 在 NetBeans 中使用单选与复选菜单项

前面的两个小节分别对 JCheckBoxMenuItem 和 JRadioButtonMenuItem 做了简要地介绍。在本小节, 将通过一个例子如何在 NetBeans 中使用这两种菜单项。本例并没有创建一个全新的程序, 而是对第 5 节给出的程序进行了改造。打开 menu 项目, 并打开其中的 DemoJMenuOne.java 文件, 按照如下步骤完成该例。

(1) 在对象观察器窗口中用鼠标右键单击 jMenuEdit 节点, 依次选择右键菜单的 “Add” / “JCheckBoxMenuItem” 选项, 向 jMenuEdit 添加一个 JCheckBoxMenuItem。将其名称修改为 jCheckBoxMenuItemBold, text 属性的值修改为 “黑体”。

(2) 再向 jMenuEdit 中添加一个名称为 jCheckBoxMenuItemItalic 的 JCheckBoxMenuItem, 并将其 text 属性的值修改为 “斜体”。

(3) 向 jMenuEdit 中添加一个 JSeparator。

(4) 在对象观察器窗口中用鼠标右键单击 jMenuEdit 节点, 依次选择右键菜单的 “Add” / “JRadioButtonMenuItem” 选项, 向 jMenuEdit 添加一个 JRadioButtonMenuItem。将其名称修改为 jRadioButtonMenuItemEighteen, text 属性的值修改为 “18 号字”, 同时将其 selected 属性设为选中状态。

(5) 按照步骤 (4) 所述, 再向 jMenuEdit 中添加两个 JRadioButtonMenuItem, 将其名称分别修改为 jRadioButtonMenuItemSixteen 和 jRadioButtonMenuItemTwenty。将 text 属性的值修改为 “16 号字” 和 “20 号字”。

(6) 向窗体中添加名称为 buttonGroupOne 的 ButtonGroup。并将上面添加的 3 个单选菜单项的 buttonGroup 属性的值均修改为 buttonGroupOne。

(7) 向 DemoJMenuOne 类添加签名为 public void dealAction(java.awt.event.ItemEvent evt) 的方法, 并将下列代码添加到该方法中:

```
if (evt.getSource() == jCheckBoxMenuItemBold) {
    //处理 jCheckBoxMenuItemBold 的事件
    if (jCheckBoxMenuItemBold.isSelected()) {
        JOptionPane.showMessageDialog(this, "你选中了黑体复选菜单!", "提示:",
            JOptionPane.INFORMATION_MESSAGE);
    } else {
        JOptionPane.showMessageDialog(this, "你取消了对黑体复选菜单的选
            择!", "提示:", JOptionPane.INFORMATION_MESSAGE);
    }
} else if (evt.getSource() == jCheckBoxMenuItemItalic) {
    //处理 jCheckBoxMenuItemItalic 的事件
```

```
if(jCheckBoxMenuItemItalic.isSelected()){
    JOptionPane.showMessageDialog(this,"你选中了斜体复选菜单!", "提示:
        ",JOptionPane.INFORMATION_MESSAGE);
}else{
    JOptionPane.showMessageDialog(this,"你取消了对斜体复选菜单的选择!
        ", "提示: ",JOptionPane.INFORMATION_MESSAGE);
}
}else if(evt.getSource()==jRadioButtonMenuItemEighteen){
    //处理 jRadioButtonMenuItemEighteen 的事件
    if(jRadioButtonMenuItemEighteen.isSelected()){
        JOptionPane.showMessageDialog(this,"“18 号字” 单选菜单被选中!", "提示:
            ",JOptionPane.INFORMATION_MESSAGE);
    }else if(evt.getSource()==jRadioButtonMenuItemSixteen){
        //处理 jRadioButtonMenuItemSixteen 的事件
        if(jRadioButtonMenuItemSixteen.isSelected()){
            JOptionPane.showMessageDialog(this,"“16 号字” 单选菜单被选中!", "提示:
                ",JOptionPane.INFORMATION_MESSAGE);
        }else if(evt.getSource()==jRadioButtonMenuItemTwenty){
            //处理 jRadioButtonMenuItemTwenty 的事件
            if(jRadioButtonMenuItemTwenty.isSelected()){
                JOptionPane.showMessageDialog(this,"“20 号字” 单选菜单被选中!", "提示:
                    ",JOptionPane.INFORMATION_MESSAGE);
            }
        }
    }
```

(8) 向 jMenuEdit 菜单中的所有 JCheckBoxMenuItem 和 JRadioButtonMenuItem 添加 ItemEvent 事件的处理方法, 并分别将下列代码添加到每个方法中:

```
this.dealAction(evt);
```

(9) 编译并运行项目, 界面与图 9-11 一致。选择 Edit 菜单, 界面如图 9-15 所示。

(10) 选中“黑体”复选框菜单, 界面如图 9-16 所示。

(11) 在如图 9-11 所示的窗口中选中“20 号字”单选菜单, 界面如图 9-17 所示。

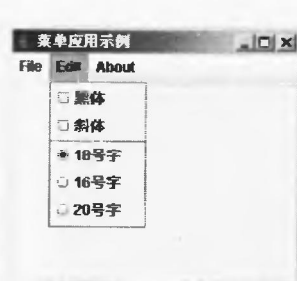


图 9-15 Edit 菜单及菜单项

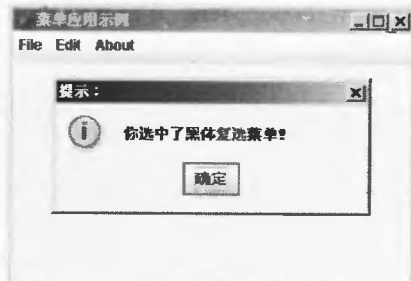


图 9-16 选中“黑体”复选框

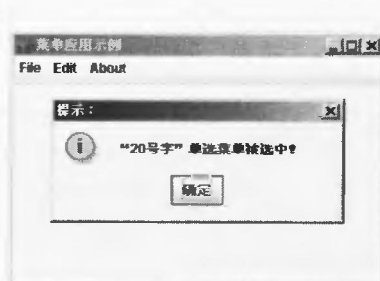


图 9-17 选中“20 号字”单选菜单

本例旨在向读者介绍如何在 NetBeans 中使用 JCheckBoxMenuItem 与 JRadioButtonMenuItem, 对于本程序的其他运行情况, 读者可以自己试验, 这里就不在全部给出了。

9.7 弹出式菜单——JPopupMenu

JPopupMenu 是一种越来越常见的菜单，本节将对 JPopupMenu 进行介绍，主要包括如下内容：

- JPopupMenu 类简介；
- 在 NetBeans 中使用 JPopupMenu。

9.7.1 JPopupMenu 类简介

JPopupMenu 类扩展了 javax.swing.JComponent，是 Swing 提供了一种菜单。与其他形式菜单不同的是，JPopupMenu 并不固定在菜单栏中，而是能够自由浮动。JPopupMenu 具有很好的环境相关（context-sensitive）特性，每一个 JPopupMenu 都与相应的控件相关联，该控件被称做调用者（invoker）。

创建 JPopupMenu 非常简单，下面代码创建了一个 JPopupMenu：

```
JPopupMenu myJPopupMenu=new JPopupMenu();
```

还可以使用 JPopupMenu 提供的另一个构造器创建带有标题的 JPopupMenu，代码如下：

```
JPopupMenu titleJPopupMenu=new JPopupMenu("弹出式菜单");
```

与 JMenu 一样，开发人员可以使用 add 方法和 insert 方法向 JPopupMenu 中添加或者插入 JMenuItem 与 JComponent。JPopupMenu 对添加到其中的每一个菜单项都赋予一个整数索引，并根据弹出式菜单的布局管理器调整菜单项显示的顺序。此外，还可以使用 addSeparator 方法添加分隔线，并且 JPopupMenu 也会为该分隔线指定一个整数索引。

开发人员通过调用弹出式菜单触发器对应的 show 方法来显示弹出式菜单，show 方法会在菜单显示之前对其 location 和 invoker 属性加以设定。

开发人员应该检查所有的 MouseEvent 事件，看其是否是弹出式菜单触发器，然后在合适的时候显示弹出式菜单，下面的 showJPopupMenu 方法在收到适当的触发器事件时就会显示弹出式菜单，代码如下：

```
public void showJPopupMenu(MouseEvent e){
    if(e.isPopupTrigger()){
        myJPopupMenu.show(invoker,e.getX(),e.getY());
    }
}
```

多种事件都可以取消弹出事件，如单击菜单项、调整调用控件的大小、移动、关闭、最小化或者最大化父窗口等。

JPopupMenu 类提供了很多方法供开发人员使用，表 9-7 给出了其中一些常用的方法及其说明。

表 9-7

JPopupMenu 类常用方法及说明

方 法	说 明
getInvoker()	返回作为此弹出菜单的“调用者”的控件
setInvoker(Component invoker)	设置弹出菜单的调用者，即弹出菜单在其中显示的控件
addPopupMenuListener(PopupMenu Listener l)	添加 PopupMenu 侦听器
removePopupMenuListener(PopupMenu Listener l)	移除 PopupMenu 侦听器
getPopupMenuListeners()	返回利用 addPopupMenuListener()添加到此 JMenuItem 的所有 PopupMenuListener 组成的数组
getLabel()	返回弹出菜单的标签
setLabel(String label)	设置弹出菜单的标签
show(Component invoker,int x,int y)	在调用者的坐标空间中的位置 X、Y 处显示弹出菜单
getComponentIndex(Component c)	返回指定控件的索引
setPopupSize(Dimension d)	使用 Dimension 对象设置弹出菜单的大小。此操作等效于 setPreferredSize(d)
setPopupSize(int width int height)	将弹出菜单的大小设置为指定的宽度和高度。此操作等效于 setPreferredSize(new Dimension(width, height))
getComponent()	返回此 JPopupMenu 控件
isPopupTrigger(MouseEvent e)	如果当前系统将 MouseEvent 视为弹出菜单触发器，则返回 true

下面给出了一个使用 JPopupMenu 的简单例子，示例代码如下：

```
package org.netbeans.swing.menu;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class DemoJPopupMenu extends JFrame {
    JMenu fileMenu;
    JPopupMenu jPopupMenuOne;
    JMenuItem openFile,closeFile,exit;
    JRadioButtonMenuItem copyFile,pasteFile;
    ButtonGroup buttonGroupOne;
    public DemoJPopupMenu() {
        jPopupMenuOne = new JPopupMenu();//创建 jPopupMenuOne 对象
        buttonGroupOne=new ButtonGroup();
        //创建文件菜单及子菜单，并将子菜单添加到文件菜单中
        fileMenu =new JMenu("文件");
        openFile=new JMenuItem("打开");
        closeFile=new JMenuItem("关闭");
        fileMenu.add(openFile);
        fileMenu.add( closeFile);
        jPopupMenuOne.add(fileMenu);//将 fileMenu 菜单添加到弹出式菜单中
        jPopupMenuOne.addSeparator();//添加分割符
        //创建单选菜单项，并添加到 ButtonGroup 对象中
        copyFile=new JRadioButtonMenuItem("复制");
```

```

pasteFile=new JRadioButtonMenuItem("粘贴");
buttonGroupOne.add(copyFile);
buttonGroupOne.add(pasteFile);
jPopupMenuOne.add(copyFile);//将 copyFile 添加到 jPopupMenuOne 中
jPopupMenuOne.add(pasteFile);//将 pasteFile 添加到 jPopupMenuOne 中
jPopupMenuOne.addSeparator();
exit=new JMenuItem("退出");
jPopupMenuOne.add(exit);//将 exit 添加到 jPopupMenuOne 中
//创建监听器对象
MouseListener popupListener = new PopupListener(jPopupMenuOne);
this.addMouseListener(popupListener);//向主窗口注册监听器
this.setTitle("弹出式菜单的简单使用");
this.setBounds(100,100,250,150);
this.setVisible(true);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public static void main(String args[]){
    new DemoJPopupMenu();
}

//添加内部类,其扩展了 MouseAdapter 类,用来处理鼠标事件
class PopupListener extends MouseAdapter {
    JPopupMenu popupMenu;
    PopupListener(JPopupMenu popupMenu) {
        this.popupMenu = popupMenu;
    }
    public void mousePressed(MouseEvent e) {
        showPopupMenu(e);
    }
    public void mouseReleased(MouseEvent e) {
        showPopupMenu(e);
    }
    private void showPopupMenu(MouseEvent e) {
        if (e.isPopupTrigger()) {//如果当前事件与鼠标事件相关,则弹出菜单
            popupMenu.show(e.getComponent(),e.getX(), e.getY());
        } //结束 if
    } //结束 showPopupMenu
} //结束内部类 PopupListener
} //结束 DemoJPopupMenu

```

编译并运行程序,界面如图 9-18 所示。在窗口中单击鼠标右键,界面如图 9-19 所示。

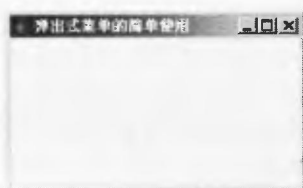


图 9-18 程序运行界面

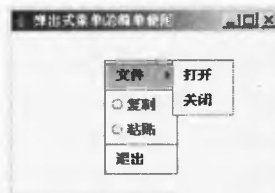


图 9-19 出现弹出式菜单

9.7.2 在 NetBeans 中使用 JPopupMenu

本节将通过一个例子介绍如何在 NetBeans 中使用 JPopupMenu。本节并没有创建一个全新的程序，而是继续改造了第 5 节中给出的程序。打开项目 Menu，并打开其中的 DemoJMenuOne.java 文件，按照如下步骤完成该例。

(1) 将 DemoJMenuOne 的布局管理器设置为 GridLayout。

(2) 向 DemoJMenuOne 中添加一个 JLabel，将其名称修改为 jLabelIcon。将 text 属性的值设置为空，并通过修改 icon 属性的值，为 jLabelIcon 添加一个图标。

(3) 向 DemoJMenuOne 中添加一个 JPopupMenu，查看对象观察器窗口，如图 9-20 所示。

(4) 从图 9-20 可以看到，将 JPopupMenu 添加到 DemoJMenuOne 后，其并没有像其他控件一样添加到 JFrame 下的某个节点上，而是成为了 Other Components 节点的子节点。这意味着其在 GUI 设计器中是不可见的。

(5) 将该 JPopupMenu 的名称修改为 jPopupMenuAboutIcon。

(6) 在对象观察器窗口中用鼠标右键单击 jPopupMenuAboutIcon 节点，依次选择右键菜单中的“Add”→“JMenuItem”选项，向 jPopupMenuAboutIcon 中添加一个 JMenuItem，将其名称修改为 jMenuItemNoIcon，并将其 text 属性的值修改为“不使用图片”。

(7) 按步骤 (6) 所述再向 jPopupMenuAboutIcon 添加一个名称为 jMenuItemNewIcon，text 属性的值为“更改图片”的 JMenuItem。此时，GUI 设计器中内容如图 9-21 所示。

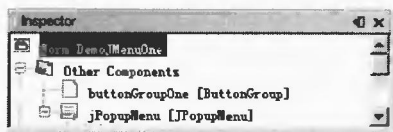


图 9-20 对象观察器窗口

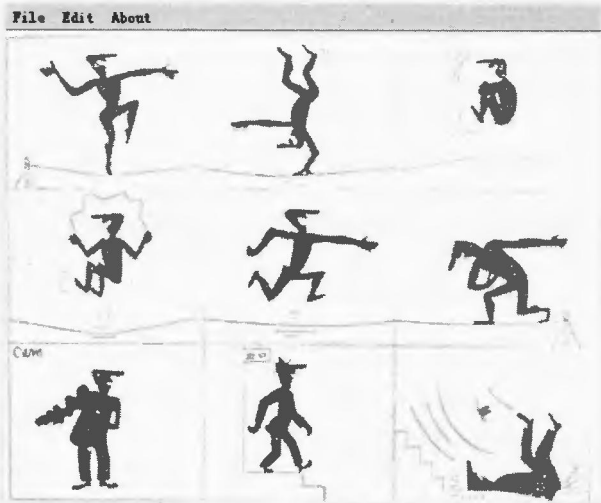


图 9-21 GUI 设计器中内容

(8) 向 DemoJMenuOne 中添加内部类。在项目窗口中选中 DemoJMenuOne 节点，单击鼠标右键，在弹出的菜单中依次选择“Inner Class”选项。出现如图 9-22 所示的窗口。

(9) 在 Name 文本框中设置要添加的内部类的名称，在 Super Class 文本框中设置要添加的内部类的父类的名称，在“Access”下拉列表框中设置内部类的访问限制修饰符。另外，还可以设置添加的内部类是否是抽象的、最终的、静态的等内容。

(10) 将要添加的类的名称修改为 PopupAdapter，父类名称为 java.awt.event.MouseAdapter，访问限制修饰符为 private，单击“OK”按钮完成设置。

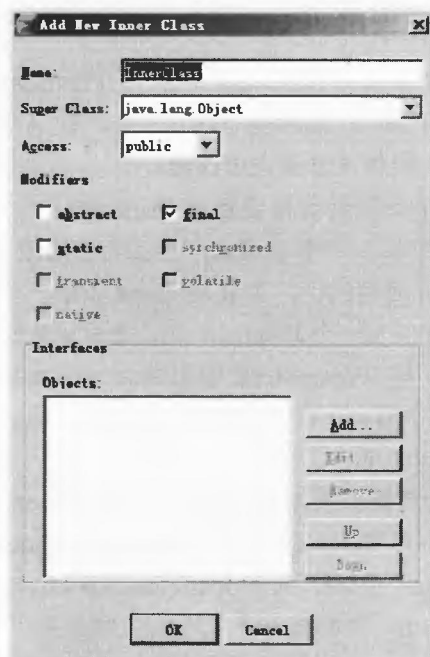


图 9-22 添加内部类窗口

(11) 将下列代码添加到 PopupAdapter 类中:

```
JPopupMenu popup;  
public PopupAdapter(JPopupMenu popupMenu) {  
    //此为内部类的构造器  
    popup = popupMenu;  
}  
public void mousePressed(MouseEvent e) {  
    //鼠标按下时调用此方法  
    showPopupMenu(e);  
}  
public void mouseReleased(MouseEvent e) {  
    //鼠标释放时调用此方法  
    showPopupMenu(e);  
}  
private void showPopupMenu(MouseEvent e) {  
    /*  
    *如果当前事件与鼠标事件相关,用产生鼠标事件的控件作为调用者,  
    *在鼠标当前位置弹出菜单  
    */  
    if (e.isPopupTrigger()) {  
        popup.show(e.getComponent(), e.getX(), e.getY());  
    }  
}
```

● 该内部类对象用来作为处理鼠标事件的监听器。

- 如果当前鼠标事件为弹出式菜单的触发器事件，则弹出弹出式菜单。

(12) 向 DemoJMenuOne 中添加名称为 iconChooser，类型为 JFileChooser，访问限制修饰符为 private 的成员变量，并将其初始值设置为 new JFileChooser()。

(13) 为 JMenuItemNoIcon 与 JMenuItemNewIcon 添加 ActionEvent 事件处理方法，并按照下面所述，向相应方法的方法体内添加代码。

- 将下列代码添加到 JMenuItemNoIconActionPerformed 方法中：

```
this.jLabelIcon.setIcon(null);
```

- 将下列代码添加到 JMenuItemNewIconActionPerformed 方法中：

```
iconChooser.showOpenDialog(this); //打开文件选择器
File iconFile=iconChooser.getSelectedFile();
if(iconFile!=null){
    //创建 ImageIcon 对象
    ImageIcon labelIcon=new ImageIcon(iconFile.toString());
    //设置 jLabelIcon 的图像
    this.jLabelIcon.setIcon(labelIcon);
}
```

(14) 将下列代码添加到 DemoJMenuOne 的构造器内，以创建 PopupAdapter 对象，并将其注册到 jLabelIcon 中。

```
PopupAdapter menuAdapter=new PopupAdapter(this.jPopupMenuAboutIcon);
this.jLabelIcon.addMouseListener(menuAdapter);
```

(15) 编译并运行程序，界面如图 9-23 所示。

(16) 在图 9-23 所示窗口中单击鼠标右键，界面如图 9-24 所示。



图 9-23 程序运行界面



图 9-24 单击鼠标右键效果图

(17) 选择“更改图片”选项，弹出文件选择器，如图 9-25 所示。

在文件选择器中选中要显示的图片，则可以在 jLabelIcon 中显示出来。本程序还有其他运行状态，这里就不全部给出了，有兴趣的读者可以自己试验这些情况。



图 9-25 选择要使用的文件

9.8 小结

本章通过具体的实例，讲述了如何使用 Swing 提供的各种菜单控件，包括 JMenu、JMenuBar、JMenuItem 以及弹出式菜单等内容。可以看到，在 NetBeans 中开发使用菜单的程序非常方便，只需要通过鼠标的拖曳即可完成大部分界面的开发，而不需要编写太多的代码。

第 10 章

Java 与图像处理

在应用程序中使用图像，可以使程序界面变得更加美观。Java 对图像处理提供了很多方便的 API。本章将对与图像处理有关的内容进行讨论，主要包括如下内容：

- Image 类；
- Icon 接口；
- JPEG 编码器的使用；
- 其他编码器介绍；
- 在 NetBeans 中开发使用图像的程序。

10.1 图像类 Image

Image 类是使用 Java 进行图像处理时经常用到的类，本节将会对 Image 类展开讨论，主要包括如下内容：

- Image 类简介；
- 在 Java 应用程序中绘制图像；
- 设置 Java 应用程序窗口的图标。

10.1.1 Image 类简介

java.awt.Image 是直接扩展自 java.lang.Object 的抽象类，是所有图像类的超类，它以特定于平台的方式获取图像。

在实际编程时，往往通过工具类的对象工厂 getImage 方法来获取实际的 Image 对象，表 10-1 中列出了常用的 getImage 方法。

表 10-1 常用的 getImage 方法

方 法	说 明	所 属 类
getImage(java.net.URL)	通过指定的地址获取图像对象	java.applet.Applet
getImage(java.net.URL, java.lang.String)	通过指定的地址获取图像对象	java.applet.Applet

续表

方 法	说 明	所 属 类
<code>getImage(java.net.URL)</code>	通过指定的地址获取图像对象	<code>java.applet.AppletContext</code>
<code>getImage(java.lang.String)</code>	通过指定的地址获取图像对象	<code>java.awt.Toolkit</code>
<code>getImage(java.net.URL)</code>	通过指定的地址获取图像对象	<code>java.awt.Toolkit</code>
<code>getImage()</code>	获取 <code>ImageIcon</code> 对象所表示图像的图像对象	<code>javax.swing.ImageIcon</code>

例如可以使用 `java.awt.Toolkit` 类的 `getImage` 方法获取一个表示 GIF 图片的 `Image` 对象, 代码如下:

```
JFrame jf=new JFrame();
Image ii=jf.getToolkit().getImage("c:\\hello.gif");
```

- 在使用 `Toolkit` 对象的 `getImage` 方法前, 首先要获取 `Toolkit` 对象, 一般可以使用 `JApplet` 或 `JFrame` 对象的 `getToolkit` 方法获取。
- 也可以使用 `Toolkit` 类中的静态方法 `getDefaultToolkit` 得到一个 `Toolkit` 对象。

`Image` 类本身提供了一些可以由其子类实现的抽象方法, 表 10-2 中给出了这些方法及其说明。

表 10-2 `Image` 类的常用方法及说明

方 法	说 明
<code>getWidth(ImageObserver observer)</code>	确定图像的宽度。如果宽度未知, 则此方法返回-1, 然后通知指定的 <code>ImageObserver</code> 对象
<code>getHeight(ImageObserver observer)</code>	确定图像的高度。如果高度未知, 则此方法返回-1, 然后通知指定的 <code>ImageObserver</code> 对象
<code>getSource()</code>	获取生成图像像素的对象
<code>getGraphics()</code>	创建供绘制闭屏图像使用的图形上下文。此方法仅供闭屏图像调用
<code>getProperty(String name, ImageObserver observer)</code>	通过名称获取此图像的属性
<code>getScaledInstance(int width,int height,int hints)</code>	创建此图像的缩放版本。返回一个新的 <code>Image</code> 对象, 默认情况下, 该对象按指定的 <code>width</code> 和 <code>height</code> 值呈现图像
<code>flush()</code>	刷新此 <code>Image</code> 对象正在使用的所有资源
<code>getCapabilities(GraphicsConfiguration gc)</code>	返回 <code>ImageCapabilities</code> 对象, 查询该对象即可了解指定 <code>GraphicsConfiguration</code> 上此 <code>Image</code> 的功能
<code>setAccelerationPriority(float priority)</code>	就加速的重要性设置此图像的提示
<code>getAccelerationPriority()</code>	返回加速优先级提示的当前值

Java 支持使用的图像文件格式不是很多, 主要包括 GIF、JPG 及 PNG 等。大多数图像操作软件都能够把图像格式转换成 Java 能够使用的格式, 当需要在 Java 中进行图像处理时, 这种转换是非常必要的。

10.1.2 在 Java 应用程序中绘制图像

上面小节介绍了如何获取 `Image` 对象。将图像读取到 Java 应用程序中以后, 即可以使用

Graphics 类中的 drawImage 方法绘制该图像，基本代码如下：

```
Image myImage= Toolkit.getDefaultToolkit().getImage("simpleimage.jpg");
g.drawImage(myImage,x,y,observer); //g 为 Graphics 类的句柄
```

Graphics 类还提供了 drawImage 的一些重载版本，表 10-3 给出了这些方法及其说明。

表 10-3 drawImage 方法及说明

方 法	说 明
drawImage(Image img,int x,int y, ImageObserver observer)	绘制指定图像中当前可用的图像。图像的左上角位于该图形上下文坐标空间的(x,y)。图像中的透明像素不影响该处已存在的像素
drawImage(Image img,int x,int y,int width, int height,ImageObserver observer)	绘制指定图像中已缩放到适合指定矩形内部的图像。图像绘制在此图形上下文坐标空间的指定矩形内部，并且如果需要，则进行缩放。透明像素不影响该处已存在的像素
drawImage(Image img,int x,int y, Color bgcolor, ImageObserver observer)	绘制指定图像中当前可用的图像。图像的左上角位于该图形上下文坐标空间的(x,y)。以指定的背景色绘制透明像素
drawImage(Image img,int x,int y,int width,int height, Color bgcolor,ImageObserver observer)	绘制指定图像中已缩放到适合指定矩形内部的图像。图像绘制在此图形上下文坐标空间的指定矩形内部，并且如果需要，则进行缩放。以指定的背景色绘制透明像素
drawImage(Image img,int dx1,int dy1,int dx2, int dy2,int sx1,int sy1, int sx2, int sy2, ImageObserver observer)	绘制当前可用的指定区域定图像的指定区域，动态地缩放图像使其符合目标绘制表面的指定区域。透明像素不影响该处已存在的像素
drawImage(Image img,int dx1,int dy1,int dx2,int dy2, int sx1,int sy1,int sx2,int sy2,Color bgcolor, ImageObserver observer)	绘制当前可用的指定图像的指定区域，动态地缩放图像使其符合目标绘制表面的指定区域

表 10-4 给出了各个 drawImage 方法中不同入口参数的说明。

表 10-4 drawImage 方法各个参数的含义

参数名称	说 明	参数名称	说 明
img	要绘制的指定图像	dx1	目标矩形第一个角的 x 坐标
dy1	目标矩形第一个角的 y 坐标	dx2	目标矩形第二个角的 x 坐标
dy2	目标矩形第二个角的 y 坐标	sx1	源矩形第一个角的 x 坐标
sy1	源矩形第一个角的 y 坐标	sx2	源矩形第二个角的 x 坐标
sy2	源矩形第二个角的 y 坐标	bgcolor	在图像非透明部分下绘制的背景色
observer	当缩放并转换了更多图像时要通知的对象		

下面给出了一个绘制图像程序的完整代码：

```
package org.netbeans.swing.image;
import java.awt.*;
import javax.swing.*;
public class DemoDrawImage extends JFrame{
```

```
Image myImage;//声明 Image 对象
String imageNameOne=new String("happy.jpg");
public DemoDrawImage() {
    //获取 Image 对象
    myImage=Toolkit.getDefaultToolkit().getImage(imageNameOne);
    //设置窗口的大小, 位置, 可见性以及标题
    this.setBounds(250,150,450,300);
    this.setTitle("Image 应用示例");
    this.validate();
    this.setVisible(true);
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
public void paint(Graphics g) {
    g.drawImage(myImage,0,0,this);//绘制图像
    g.dispose();
}
public static void main(String args[]){
    new DemoDrawImage();
}
}
```

编译并运行程序, 结果如图 10-1 所示。



图 10-1 程序运行界面

10.1.3 设置 Java 应用程序窗口的图标

使用 Java 开发的应用程序, 其窗口默认的图标是 Java 的标志——一杯飘香的咖啡。开发人员可以使用特定的 Image 对象作为窗口的图标, 下面给出了使用特定 Image 对象作为窗口图标的完整代码:

提示 一个正式发布的程序应该设置窗口图标, 而不应该使用默认图标。


```
package org.netbeans.swing.image;
import java.awt.*;
import javax.swing.*;
public class DemoTitleIcon extends JFrame{
    Image iconImage;
    String imageName= "ball.jpg";
    public DemoTitleIcon(){
        iconImage=Toolkit.getDefaultToolkit().getImage(imageName);
        //JFrame 对象的 setIconImage 方法用来设置窗体的图标
        this.setIconImage(iconImage);
        this.setBounds(300,200,250,200);
        this.setTitle("设置窗口的图标");
        this.setVisible(true);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    public static void main(String args[]){
        new DemoTitleIcon();
    }
}
```

编译并运行程序，结果如图 10-2 所示。



图 10-2 设置窗口图标

10.2 Swing 图标 ImageIcon

Swing 中引入了图标的概念，它能够被很多种控件方便地使用。本节将介绍 Swing 中与图标有关的类和接口，主要包括如下内容：

- Icon 接口；
- ImageIcon 类。

10.2.1 Icon 接口

Icon 接口非常简单，其中仅有 `getIconWidth`、`getIconHeight` 和 `paintIcon` 3 个方法。前两个方法用来确定图标的大小，而后一个方法用来在指定的控件上绘制图像。

开发人员可以随意地实现 Icon 接口，这就意味着图标不一定是 JPG、PNG 或者 GIF 等通用格式，而可以是任何格式。下面是一个实现了 Icon 接口的类的代码，其将 3 个逐渐变暗的椭圆作为图标的内容。

```
package org.netbeans.swing.image;
import java.awt.*;
```

```

import javax.swing.*;
public class DemoRectIcon implements Icon{
    private int width,height;//声明图标的长、宽属性
    public DemoRectIcon(int width,int height) {
        this.width=width;
        this.height=height;
    }
    public void paintIcon(Component comp,Graphics g,int x,int y) {
        g.setColor(Color.black);//设置画笔颜色为黑色
        g.fill3DRect(x,y,width,height,true);//绘制 3D 矩形
        int span=width/3;
        Color c=Color.white;
        for(int i=0;i<3;i++)
        {
            g.setColor(c);//设置画笔颜色
            g.fillOval(i*span+x,2+y,span-4,height-4); //绘制椭圆
            c=c.darker(); //将颜色调暗
        }
    }
    //返回图标宽度
    public int getIconWidth(){return this.width;}
    public int getIconHeight(){return this.height;}
}

```

下面代码使用了前面创建的图标。

```

package org.netbeans.swing.image;
import java.awt.*;
import javax.swing.*;
public class UseRectIcon extends JFrame{
    JLabel showLabel;
    public UseRectIcon() {
        showLabel=new JLabel(); //创建标签控件
        showLabel.setHorizontalAlignment(SwingConstants.CENTER); //设置对齐方式
        showLabel.setIcon(new DemoRectIcon(120,80)); //使用 DemoRectIcon 类
        创建图标
        this.add(showLabel);
        this.setBounds(100,100,250,150);
        this.setTitle("使用 Icon 示例");
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置窗体默
        认关闭动作
    }
    public static void main(String args[]){
        new UseRectIcon();
    }
}

```

编译并运行项目，结果如图 10-3 所示。

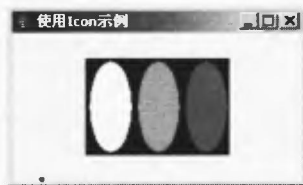


图 10-3 程序运行界面

10.2.2 ImageIcon 类

`javax.swing.ImageIcon` 类是 `Icon` 接口的具体实现。`ImageIcon` 可以显示动态图像 GIF89a，并且支持同步图像加载（即在返回之前完全下载图像），这些使得 `ImageIcon` 具有非常强大的功能。创建 `ImageIcon` 对象十分简单，下面代码给出了创建 `ImageIcon` 对象的方法。

```
ImageIcon myImageIcon=new ImageIcon("tupian.jpg");
```

`ImageIcon` 类提供了 9 个构造器，表 10-5 列出了常用的 4 个构造器及其说明。

表 10-5 ImageIcon 类的常用构造器及说明

构造器	说明
<code>ImageIcon()</code>	创建一个不包含任何内容的 <code>ImageIcon</code>
<code>ImageIcon(String filename)</code>	根据指定的文件创建一个 <code>ImageIcon</code>
<code>ImageIcon(URL location)</code>	根据指定的 URL 创建一个 <code>ImageIcon</code>
<code>ImageIcon(Image image)</code>	使用给定的 <code>Image</code> 对象创建 <code>ImageIcon</code>

`ImageIcon` 类提供了一些方法供开发人员使用，表 10-6 列出了常用的方法及说明。

表 10-6 ImageIcon 类的常用方法及说明

方法	说明
<code>getImage()</code>	返回此图标的 <code>Image</code> 对象
<code>setImage(Image image)</code>	设置由此图标显示的图像
<code>getDescription()</code>	获得图像的描述
<code>setDescription(String description)</code>	设置图像的描述
<code>setImageObserver(ImageObserver observer)</code>	设置图像的图像观察者。如果 <code>ImageIcon</code> 包含一个动画 GIF，则设置此属性，而观察者获得更新其显示的通知
<code>ImageObserver getImageObserver()</code>	返回图像的图像观察者

提示 与绝大多数 Swing 类一样，`ImageIcon` 类是可序列化的。但 `ImageIcon` 使用的 `java.awt.image` 是不可序列化的，这会影响 `ImageIcon` 对象序列化。

下面给出了使用了 `ImageIcon` 程序的完整代码：

```
package org.netbeans.swing.image;
import java.awt.*;
import java.awt.event.*;
```

```
import javax.swing.*;
import javax.swing.border.*;
public class ShowLanternSlide extends JFrame implements ActionListener{
    JLabel imageLabel;
    JButton startButton;
    JPanel southPanel;
    String [] iconFileName;
    public ShowLanternSlide() {
        imageLabel=new JLabel();
        imageLabel.setBorder(new TitledBorder("")); //设置 imageLabel 的边框
        startButton=new JButton("播放幻灯片");
        startButton.addActionListener(this);
        southPanel=new JPanel();
        imageLabel.setHorizontalAlignment(SwingConstants.CENTER);
        southPanel.add(startButton);
        this.add(imageLabel,BorderLayout.CENTER);
        this.add(southPanel,BorderLayout.SOUTH);
        this.setBounds(100,100,250,150);
        this.setTitle("使用 ImageIcon 示例");
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String args[]){
        new ShowLanternSlide();
    }

    //该方法负责播放幻灯片
    public void showImageIcon(){
        File tempFile=new File(".");//创建文件
        String [] allFileName=tempFile.list(); //获取当前目录所有文件的名称
        int allFileNameSize=allFileName.length; //获取当前目录文件的个数
        iconFileName=new String[allFileNameSize]; //初始化 iconFileName 数组
        int iconFileSize=0;
        for(int i=0;i<allFileNameSize;i++){
            if(this.accept(allFileName[i])){
                //将 allFileName 中符合要求的文件名称存到 iconFileName 中
                iconFileName[iconFileSize]= allFileName[i];
                iconFileSize++;
            }
        }

        Thread myThread=new Thread(){
            public void run(){
                for(int i=0;i<iconFileName.length;i++){
                    ShowLanternSlide.this.imageLabel.setIcon(new
                        ImageIcon(ShowLanternSlide.this.iconFileName[i]));
                    try{
                        Thread.sleep(3000); //线程睡眠 3000 毫秒
                    }catch(Exception e){

```

```
        e.printStackTrace();
    } //end catch
    } //end for
    } //run
} //end new Thread
myThread.start(); //启动线程
}
public boolean accept(String fileName){
    boolean flag=false;
    if(fileName.toLowerCase().endsWith(".jpg")) {
        flag=true; //如果传入文件的名称为以.jpg 结尾, 返回 true
    } else if(fileName.toLowerCase().endsWith(".jpeg")) {
        flag=true; //如果传入文件的名称为以.jpeg 结尾, 返回 true
    } else if(fileName.toLowerCase().endsWith(".gif")) {
        flag=true; //如果传入文件的名称为以.gif 结尾, 返回 true
    } else if(fileName.toLowerCase().endsWith(".png")) {
        flag=true; //如果传入文件的名称为以.png 结尾, 返回 true
    }
    return flag;
}
public void actionPerformed(ActionEvent e){
    if(e.getSource()==startButton)
        this.showImageIcon();
}
}
```

编译并运行程序, 结果如图 10-4 所示。



图 10-4 程序运行界面

单击“播放幻灯片”按钮, 窗口中会将当前目录的图像文件以幻灯片的形式播放, 图 10-5 给出了播放过程中的几个画面。



图 10-5 播放幻灯片的过程

本程序仅仅是将当前目录中的图像文件以幻灯片的形式顺序播放出来, 还可以扩展该程

序, 添加更多的功能, 如查看上一幅图像、查看下一副图像、暂停播放幻灯片、继续播放幻灯片等。有兴趣的读者可以自己完成这些功能, 这里不再详细介绍了。

10.3 图像处理的高级应用——JPEG 编码器

前面小节介绍了 Image 类与 Icon 接口的使用。在 Java 中不仅可以使⽤已有的图像, 还可以将程序中的图像保存到文件中。如果需要将程序中的图像保存到文件中, 并且能够被其他的普通图片查看器所查看, 则需要将图像保存为 JPEG 或其他通用的图像格式。

JPEG 是静态数字图像压缩编码方法国际标准, JDK 中 com.sun.image.codec.jpeg 包内的 JPEGImageEncoder 接口提供了将程序中的图像编码成 JPEG 格式的方法。

JPEGImageEncoder 用来将缓冲区中的图像数据编码为 JPEG 数据流, 使用该接口需要提供 Raster 或者 BufferedImage 对象。以下代码介绍了如何使用 JPEGImageEncoder 进行编码。

```
FileOutputStream fos=new FileOutputStream("example.jpg");//创建输出流
JPEGImageEncoder encoder=JPEGCodec.createJPEGEncoder(fos);
encoder.encode(bufIma);// bufIma 为已有的 BufferedImage 对象
fos.flush();
fos.close();
```

可以看到, 使用 JPEGImageEncoder 对图像进行编码非常简单。首先创建一个输出流, 接着使用 JPEGCodec 类的静态方法 createJPEGEncoder 把编码器连接到输出流, 然后使用 encode 方法对已有的 BufferedImage 对象进行编码, 最后输出该流即可。

JPEGCodec 是 com.sun.image.codec.jpeg 包提供的对象工厂类, 其中主要包含 createJPEGEncoder 和 getDefaultJPEGEncodeParam 两个方法, 前者用来返回 JPEGImageEncoder, 后者用来返回 JPEGEncodeParam。每个方法又提供了不同的重载版本, 详细情况请读者查阅相关的 API。



提示

JPEGEncodeParam 封装了控制 JPEG 数据流编码的必要信息, 用到该接口的读者可以查阅相关的 API, 这里不再详细讲述。

10.4 其他编码器介绍

随着技术的发展, 图像的压缩格式也越来越多, 但是标准的 JDK 中只提供了一个 JPEG 格式的编码器, 这远远不能满足开发的需要。

在实际开发中, 如果现有的编码器不能满足需要, 有两种选择, 一个是自己动手编写需要的编码器, 另一个是寻找合适的第三方编码器。自己动手编写编码器有一定的难度, 这里不做介绍, 本节将介绍由 Acme 公司推出的第三方 GIF 编码器 GifEncoder, 并通过一个例子来介绍其使用方法。本节主要包括如下内容:

- GifEncoder 简介。
- 使用 GifEncoder 编码器程序的界面设计。
- 使用 GifEncoder 编码器程序的功能代码的开发。

10.4.1 GifEncoder 简介

GifEncoder 是 Acme 推出的免费 GIF 编码器，其主要用来将 JPG 图像转成 GIF。可以从以下地址下载该编码器：http://iron.gps.caltech.edu/trinet_tpp_doc/v1.5/download/openmap.jar。

下载该编码器以后，需要将 openmap.jar 文件添加到类路径中，并且在程序中使用“import Acme.JPM.Encoders.GifEncoder”语句导入 GifEncoder 类，之后则可以在程序中使用 GifEncoder 类了。以下代码创建了一个 GifEncoder 对象：

```
GifEncoder myGifEncoder=new GifEncoder(myImage ,out);
```

其中，myImage 与 out 分别是已有的 Image 与 OutputStream 对象。

GifEncoder 提供了 4 个构造器，表 10-7 列出了这几个构造器及其说明。

表 10-7 GifEncoder 构造器及说明

构造器	说 明
GifEncoder(Image img,OutputStream out)	使用给定的 Image 对象和输出流创建 GifEncoder 对象
GifEncoder(Image img,OutputStream out, boolean interlace)	使用给定的 Image 对象，输出流以及扫描方式创建 GifEncoder 对象，interlace 用来指定是否使用隔行扫描
GifEncoder(ImageProducer prod, OutputStream out)	使用给定的 ImageProducer 对象以及输出流创建 GifEncoder
GifEncoder(ImageProducer prod,OutputStream out, boolean interlace)	使用给定的 ImageProducer 对象，输出流以及扫描方式创建 GifEncoder

创建 GifEncoder 对象以后即可以使用 encode 方法对图像进行编码：

```
myGifEncoder.encode();
out.flush();//刷新输出流
```

GifEncoder 类还提供了另外一些方法供开发人员使用，有兴趣的读者可以在以下地址找到这些方法的详细说明：<http://www.acme.com/java/software>。

10.4.2 使用 GifEncoder 编码器程序的界面设计

本程序的界面设计比较简单，启动 NetBeans，按照如下步骤完成界面设计。

- (1) 创建一个名称为 image 的项目，将其主类名称设置为 org.netbeans.swing.image.Demo。
- (2) 向 image 中添加一个使用“JFrame Form”模版创建的名称为 UseGifEncoder 的类。
- (3) 向 UseGifEncoder 中添加一个 JMenuBar，将其名称修改为 jMenuBarGlobal。并将其自动生成的菜单名称修改为 jMenuFile，text 属性的值设置为“文件”。
- (4) 向 jMenuFile 中添加 3 个 JMenuItem，并按照表 10-8 修改属性。

表 10-8 属性与值对照表

名 称	text 属性值	加速键
jMenuItemOpenFile	打开文件	Ctrl+O
jMenuItemSaveFile	保存文件	Ctrl+S
jMenuItemExit	退出	Ctrl+E

(5) 向 UseGifEncoder 中添加一个 JScrollPane，并将其名称修改为 jScrollPaneGlobal。

(6) 向 UseGifEncoder 中添加一个 JLabel，将其名称修改为 jLabelImage，并将其 text 属性的值设置为空。此时，GUI 设计器中内容如图 10-6 所示。

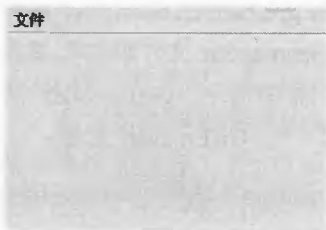


图 10-6 GUI 设计器中内容

10.4.3 使用 GifEncoder 编码器程序的功能代码的开发

按照如下步骤完成功能代码的开发。

(1) 首先需要将包含 GifEncoder 的 jar 文件添加到 NetBeans 的类路径中：在项目窗口中右键单击“Libraries”节点，选择菜单中的“Add JAR/Folder”，出现如图 10-7 所示窗口。

(2) 在图 10-7 所示窗口中选择要添加的 jar 文件，单击“打开”按钮完成添加。此时如果查看项目窗口，会发现“Libraries”节点会多出一个名称为 openmap.jar 的子节点，表明添加成功，如图 10-8 所示。

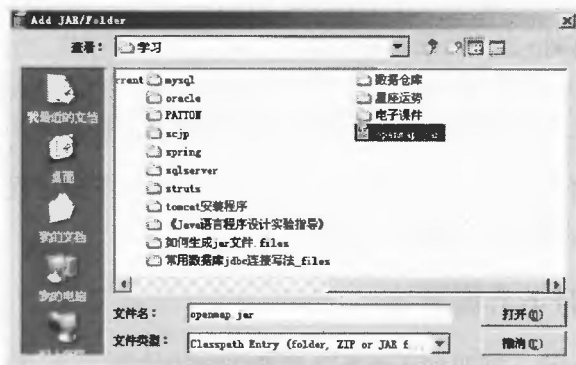


图 10-7 选择 jar 文件

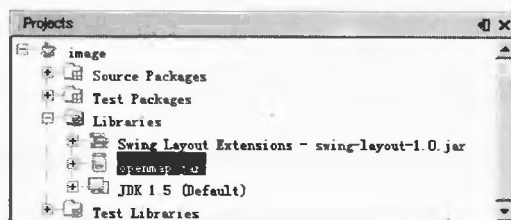


图 10-8 添加 jar 文件后项目窗口

(3) 按表 10-9 所示向 UseGifEncoder 中添加成员变量。

表 10-9 添加成员变量属性

变量名称	变量类型	访问限制修饰符	初始值
encoder	GifEncoder	private	null
fileChooser	JFileChooser	private	new JFileChooser()
selectFile	File	private	null

(4) 向 UseGifEncoder 中添加签名为 private void openImageFile 的方法，该方法主要用来获得打开图像的名称，并将该图像在 jLabelImage 上显示。将下列代码添加到该方法的方法体内。

```

fileChooser.showOpenDialog(this);
openedFile=fileChooser.getSelectedFile();
if(openedFile==null)
    return;
else{
    this.jLabelImage.setIcon(new ImageIcon(openedFile.getPath()));
}

```

(5) 向 UseGifEncoder 中添加签名为 saveImageAsGif 的方法, 该方法用来将当前打开的文件保存为 GIF 格式, 将下列代码添加到该方法的方法代码中:

```

try {
    int saved=fileChooser.showSaveDialog(this);
    if(saved==JFileChooser.APPROVE_OPTION) {
        if(this.openedFile!=null){//当前已经打开了文件
            File saveFileName=fileChooser.getSelectedFile();
            //获取要保存文件的名字
            String fileName=new String(saveFileName.getPath()+".gif");
            //创建一个输出流
            FileOutputStream fileOutPut=new FileOutputStream(fileName);
            //使用打开的文件名称获取一个输入流
            InputStream ins=new FileInputStream(this.openedFile.
                toString());
            //使用 ImageIO 获取 BufferedImage 对象
            BufferedImage srcImage = ImageIO.read(ins);
            //使用 BufferedImage 与 InputStream 对象创建 GifEncoder 对象
            encoder = new GifEncoder(srcImage, fileOutPut);
            encoder.encode();//使用 encode 方法进行编码
            fileOutPut.flush();//刷新输出流
            fileOutPut.close();//关闭输出流
            ins.close();//关闭输入流
        }
    }
} catch(Exception ea) {
    ea.printStackTrace();
}

```

(6) 为 UseGifEncoder 的 3 个菜单项添加 ActionEvent 事件的处理方法, 并按照下面所述向相应方法的方法体内添加代码。

- 向 jMenuItemOpenFileActionPerformed 方法的方法体内添加如下代码:

```
this.openImageFile();
```

- 向 jMenuItemSaveFileActionPerformed 方法的方法体内添加如下代码:

```
this.saveImageAsGif();
```

- 向 jMenuItemExitActionPerformed 方法的方法体内添加如下代码:

```
System.exit(0);
```

(7) 将下列代码添加到 Demo 的 `public static void main(String args[])` 方法的代码中:

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new UseGifEncoder ().setVisible(true);  
    }  
});
```

(8) 编译并运行项目, 结果如图 10-9 所示。按下“Ctrl+O”快捷键打开文件选择器, 界面如图 10-10 所示。

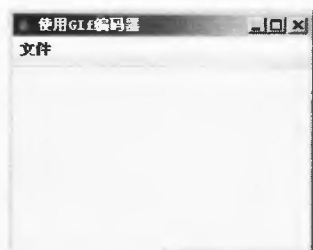


图 10-9 程序运行界面

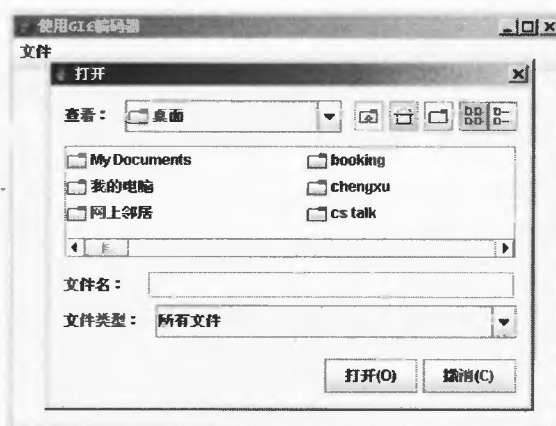


图 10-10 打开文件对话框

(9) 选中要打开的图像文件, 单击“打开”按钮打开该文件。此时界面如图 10-11 所示。



图 10-11 打开图像文件

(10) 按下 Ctrl+S 快捷键打开文件保存对话框, 选择相应的保存路径, 并将文件的名称设置为 `happying` 后保存文件。

(11) 此时查看保存文件的目录, 会发现在该目录下确实存在一个名称为 `happying.gif` 的文件。查看该文件, 会发现其与前面打开的文件基本一致 (将 jpg 格式转换为 gif 格式, 可能

会导致颜色丢失)。

10.5 在 NetBeans 中进行图像处理的案例

使用 NetBeans 可使图像处理更加简单,本节将通过一个完整的项目介绍如何在 NetBeans 中进行图像处理。

10.5.1 界面设计

本项目的界面比较简单,启动 NetBeans,按照如下步骤完成界面设计。

- (1) 打开前面创建的项目 image。
- (2) 向 image 中添加一个通过“JFrame Form”模版创建的名称为 DemoImage 的类。
- (3) 向 DemoImage 中添加一个 JMenuBar,将其名称修改为 jMenuBarAll,并将其自动生成的菜单名称修改为 jMenuItem, text 属性的值设置为“操作”。
- (4) 向 jMenuItem 中添加 4 个 JMenuItem,并按照如表 10-10 所示修改属性。

表 10-10

属性与值对照表

名 称	text 属性值	加 速 键	初 始 状 态
jMenuItemCapture	开始截图	Ctrl+Shift+A	可操作
jMenuItemSaveAs	文件另存为	Ctrl+S	不可操作
jMenuItemNolcon	清空内容	Ctrl+Shift+ S	不可操作
jMenuItemExit	退出	Ctrl+E	可操作

- (5) 向 DemoImage 中添加一个 JScrollPane,并将其名称修改为 jScrollPaneGlobal。
- (6) 向 jScrollPaneGlobal 中添加一个 JLabel,将其名称修改为 jLabelShowIcon,并将其 text 属性的值设置为空。此时,GUI 设计器内容如图 10-12 所示。

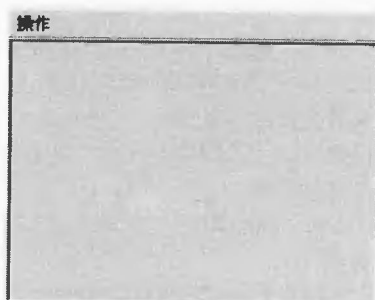


图 10-12 GUI 设计器内容

10.5.2 功能代码的开发

按照如下步骤完成功能代码的开发。

- (1) 按表 10-11 所示向 DemoImage 中添加成员变量。
- (2) 向 DemoImage 中添加签名为 `private Image createImage` 的方法。将下列代码添加到该方法的方法代码中。

表 10-11


添加成员变量属性

变量名称	变量类型	访问限制修饰符	初始值
tempImage	Image	private	null
encoder	JPEGImageEncoder	private	null
fileChooser	JFileChooser	private	new JFileChooser()

```

try { //截图代码开始
    Image tempLocalImage=null;
    Robot robot=new Robot(); //创建 Robot 对象
    Dimension dimension= Toolkit.getDefaultToolkit().getScreenSize();
    //获取屏幕大小
    Rectangle scrRect=new Rectangle(0,0,dimension.width,dimension.
height);
    //截取指定的矩形区域内图像
    tempLocalImage =robot.createScreenCapture(scrRect);
    //截图代码结束
    return tempLocalImage; //返回截取的图像
} catch (AWTException ex) {
    ex.printStackTrace();
}
return null;

```

 **说明** 此方法的功能为抓取当前桌面的图像并将其作为 Image 对象返回。

(3) 向 DemoImage 中添加签名为 public void saveImage 的方法, 然后将下列代码添加到该方法的方法体内。

```

try{
    //打开文件保存对话框
    int saved=fileChooser.showSaveDialog(this);
    if(saved==JFileChooser.APPROVE_OPTION){
        //获取设置的文件名(包括路径)
        File saveFileName=fileChooser.getSelectedFile();
        //获取要保存文件的名称
        String fileName=new String(saveFileName.getPath()+".jpg");
        //创建一个输出流
        FileOutputStream fileOutPut =new FileOutputStream(fileName);
        //把输出流用 JPEG 编码器进行包裹, 即把输出流连接到编码器
        encoder=JPEGCodec.createJPEGEncoder(fileOutPut);
        //对 BufferedImage 对象进行编码
        encoder.encode((BufferedImage) tempImage);
        fileOutPut.flush();
        fileOutPut.close();
    }
}
catch(Exception ea) {

```

```
ea.printStackTrace();  
}
```

说明 该方法用来将 `createImage` 方法创建的对象保存到用户指定的文件中。

(4) 为 `jMenuItem` 的 4 个菜单项添加 `ActionEvent` 事件的处理方法, 并按照下面所述向相应方法的方法体内添加代码。

● 向 `jMenuItemCaptureActionPerformed` 方法的方法体内添加如下代码:

```
Image captureImage=this.createImage();//截取当前屏幕图像  
//将图像在 jLabelShowIcon 中显示  
jLabelShowIcon.setImage(new ImageIcon(tempImage=this.createImage()));  
this jMenuItemSaveAs.setEnabled(true);  
this jMenuItemNoIcon.setEnabled(true);  
this jMenuItemCapture.setEnabled(false);
```

● 向 `jMenuItemSaveAsActionPerformed` 方法的方法体内添加如下代码:

```
this.saveImage();//调用 saveIcon 方法保存当前图像
```

● 向 `jMenuItemNoIconActionPerformed` 方法的方法体内添加如下代码:

```
this.jLabelShowIcon.setIcon(null);//将 jLabelShowIcon 中的图像清空  
this jMenuItemSaveAs.setEnabled(false);  
this jMenuItemNoIcon.setEnabled(false);  
this jMenuItemCapture.setEnabled(true);
```

● 向 `jMenuItemExitActionPerformed` 方法的方法体内添加如下代码:

```
System.exit(0);//推出程序
```

(5) 将下列代码添加到 `Demo` 的 `public static void main(String args[])` 方法的方法体内。

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new DemoImage().setVisible(true);  
    }  
});
```

(6) 编译并运行程序, 结果如图 10-13 所示。

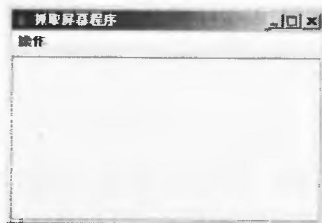


图 10-13 程序运行界面

(7) 此时, 按下抓图加速键 `Ctrl+Shift+A` 抓图, 界面如图 10-14 所示。



NetBeans IDE 5.0 - auge

File Edit View Navigate Source Refactor Build Run CVS Tools Window Help

Project

保存

保存: 我的电脑

3.5 软盘 (A:) 本地磁盘 (E:) SOFTWARE (G:)

2000 os (C:) 共享文档 MEDIA AND GAME (I:)

2000 database (D:) JAVA (F:) xp database (L:)

文件名:

文件类型: 所有文件

保存(S) 取消(C)

(9) 选择文件保存目录并填写文件名称后, 单击“保存”按钮。找到文件的保存目录, 会发现确实增加了一个.jpg 格式的文件。打开该文件, 会发现其内容与图 10-14 所示一致。

另外, 还可以通过选择菜单中的“清空内容”选项, 将窗口的内容设置为空。对于本程序的其他运行状态, 有兴趣的读者可以进行做实验, 这里就不详细说明了。

10.6 小结

本章结合具体的实例讲述了 Java 中与图形图像有关的知识，包括 Icon 接口、ImageIcon 类以及 Image 类，介绍了 Java 扩展包中提供的 JPEG 编码器，并介绍了如何使用第三方 GIF 编码器进行项目开发。在本章的最后还结合具体的实例介绍了如何使用 JPEG 编码器进行项目开发。

第 11 章

树状列表——JTree

树是一种非常重要的数据组织形式，任何具有父子关系的数据都可以由树状结构来表示。使用树能够更直观地显示分级信息，使信息的遍历和管理更加方便。Swing 提供了 JTree 控件，使用它可以很方便地创建具有树状结构的视图。在本节中将对与树有关的内容展开讨论，主要包括如下内容：

- 与树有关的一些概念；
- JTree；
- DefaultTreeModel；
- DefaultMutableTreeNode；
- TreePath；
- 树的事件；
- DefaultTreeCellRenderer；
- 在 NetBeans 中开发树状结构的程序实例。

11.1 与树有关的一些概念

在开始对 Swing 中的树进行介绍之前，先介绍一些与树有关的概念。图 11-1 显示了一棵简单的树，其节点由 A、B、C、D、E、F、G、H 7 个字母表示。

- 节点：树中的任意条目。A~H 都是节点。
- 根节点：树的顶级条目。根节点不能有父节点，一个树结构只能有一个根节点，本例的根节点是 A。
- 子节点：给定节点的下层节点，如 D 是 B 的子节点，F 是 C 的子节点。

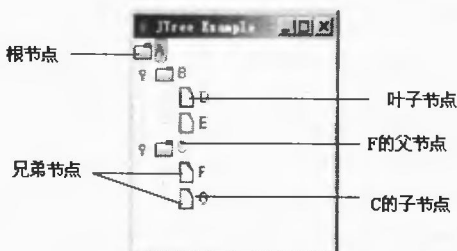


图 11-1 简单树

- 父节点：给定节点的上层节点，如 B 是 E 的父节点，C 是 G 的父节点。
- 兄弟节点：与某个节点有同一个父节点的节点，如 D 与 E，F 与 G 均是兄弟节点。
- 级：节点到根节点的距离，最大级又称为树的深度。如 B 是 1 级，F 是 2 级。
- 路径：节点到节点经过的节点序列。一般情况下，路径是指根节点到其他节点经过的节点序列。
- 折叠：直观表示树时，如果看不到某个节点的子节点，则该节点处于折叠状态。
- 展开：直观表示树时，如果能看到某个节点的子节点，则该节点就处于展开状态。
- 行：树的当前显示情况下，路径映射到屏幕上的相应行数。A 始终处于 0 行，其他的节点可能会根据节点的展开与折叠情况映射到不同的行。

11.2 JTree 介绍

JTree 是开发具有树状结构的程序时经常使用的一种控件，本节将会对 JTree 控件进行介绍，主要包括如下内容：

- JTree 类简介。
- JTree 类的方法说明。
- 使用 JTree 的程序实例。

11.2.1 JTree 类构造器说明

javax.swing.JTree 扩展了 javax.swing.JComponent，它是一个标准的 Swing 控件。与其他大多数 Swing 控件一样，JTree 采用了 MVC 模式。使用 JTree 类创建的仅是数据的视图，而没有包含真正的数据，其数据一般由 TreeModel 管理。下面的代码创建了一个 JTree 对象：

```
JTree myJTree=new JTree();
```

此时创建的 JTree 中不包含任何节点，开发人员还可以使用 JTree 的其他构造器创建具有初始内容的 JTree 对象，表 11-1 列出了这些构造器及其说明。

表 11-1 JTree 类常用构造器及说明

构造器	说 明
JTree(Object[] value)	返回 JTree，指定数组的每个元素作为不被显示的新根节点的子节点。默认情况下，树可以将叶节点定义为不带子节点的任何节点
JTree(Vector value)	返回 JTree，指定 Vector 的每个元素作为不被显示的新根节点的子节点
JTree(Hashtable value)	返回从 Hashtable 创建的 JTree，它不显示根。HashTable 中每个键/值对的半值都成为新根节点的子节点
JTree(TreeNode root)	返回一个 JTree，指定的 TreeNode 作为其根，它显示根节点
JTree(TreeNode root, boolean asksAllowsChildren)	返回一个 JTree，指定的 TreeNode 作为其根，它用指定的方式显示根节点，并确定节点是否为叶节点
JTree(TreeModel newModel)	返回 JTree 的一个实例，它显示根节点，使用指定的数据模型创建树

以下代码使用 Vector 作为数据模型创建了具有 3 个节点的 JTree。

```
Vector modelVector=new Vector();
```

```
modelVector.add("节点 1");
modelVector.add("节点 2");
modelVector.add("节点 3");
JTree myJTree=new JTree(modelVector);
```

❗注意 读者还可以使用其他构造器创建 JTree，这里就不再给出具体的示例了。

11.2.2 JTree 类的方法说明

JTree 类提供了大量的方法供开发人员使用，如可以使用 setEditable 方法设置树中节点的可编辑性，代码如下：

```
myJTree.setEditable(true); //入口参数为 true 表示设置节点为可编辑的
```

说明 使用上述方法后，用户就可以对节点内容进行编辑。方法是在该节点上连续单击鼠标左键三次，此时可以修改该节点的内容，最后按下“ENTER”键完成。

还可以使用 getRowCount 方法获取树中当前显示的行数，代码如下：

```
myJTree.getRowCount();
```

除了上面给出的两个方法，JTree 类还提供了大量的方法供开发人员使用，表 11-2~表 11-4 列出了其中一些常用的方法及说明。

表 11-2 设置方法及说明

方 法	说 明
isRootVisible()	如果显示树的根节点，则返回 true
setRootVisible(boolean rootVisible)	确定 TreeModel 的根节点是否可见
setShowsRootHandles(boolean newValue)	设置 showsRootHandles 属性的值，它指定是否应显示节点句柄。此属性的默认值取决于用于创建 JTree 的构造方法
getShowsRootHandles()	返回 showsRootHandles 属性的值
setScrollsOnExpand(boolean newValue)	设置 scrollsOnExpand 属性，它确定树是否可以滚动显示以前隐藏的子节点。如果此属性为 true（默认值），则当节点展开时，树可以使用滚动来显示该节点的尽可能多的子节点
getScrollsOnExpand()	返回 scrollsOnExpand 属性的值
setToggleClickCount(int clickCount)	设置展开与折叠树节点的鼠标单击次数，默认值为两次
getToggleClickCount()	返回展开或关闭节点所需的鼠标单击数
setExpandsSelectedPaths(boolean newValue)	配置 expandsSelectedPaths 属性。如果为 true，则在任何时候通过 TreeSelectionModel 或 Tree 提供的覆盖方法更改选择时，将展开 TreePath 的父路径，使它们可见。如果为 false，当选择更改时，节点的父节点不可见
getExpandsSelectedPaths()	返回 expandsSelectedPaths 属性
setVisibleRowCount(int newCount)	设置要显示的行数。仅当树包含在 JScrollPane 中时才工作，并将调整首选大小和该滚动窗格的大小
getVisibleRowCount()	返回显示区域中显示的行数

续表

方 法	说 明
<code>setExpandedState(TreePath path,boolean state)</code>	设置此 JTree 的展开状态。如果 state 为 true, 则 path 的所有父路径和子路径都被标记为展开。如果 state 为 false, 则 path 的所有父路径被标记为展开, 但是 path 本身被标记为折叠
<code>setCellRenderer(TreeCellRenderer x)</code>	设置用于绘制树节点的绘制器对象

表 11-3

选择方法及说明

方 法	说 明
<code>setSelectionPath(TreePath path)</code>	选择指定路径标识的节点。如果隐藏该路径的任何组件(在折叠节点之下), 并且 <code>getExpandsSelectedPaths</code> 为 true, 则可查看
<code>setSelectionPaths(TreePath[] paths)</code>	选择由指定的路径数组标识的节点
<code>setSelectionRow(int row)</code>	选择显示的指定行的节点
<code>setSelectionRows(int[] rows)</code>	选择与显示的每个指定行对应的节点
<code>addSelectionPath(TreePath path)</code>	将由指定的 TreePath 标识的节点添加到当前选择
<code>addSelectionRow(int row)</code>	将指定行处的路径添加到当前选择
<code>addSelectionRows(int[] rows)</code>	将每个指定行处的路径添加到当前选择
<code>getSelectionPath()</code>	返回首选节点的路径
<code>getSelectionPaths()</code>	返回所有选择值的路径
<code>getSelectionCount()</code>	返回选择的节点数
<code>getMinSelectionRow()</code>	获取选择的第一行
<code>getMaxSelectionRow()</code>	返回选择的最后一行
<code>isRowSelected(int row)</code>	如果选择了行标识的节点, 则返回 true
<code>setSelectionModel(TreeSelectionModel selectionModel)</code>	设置树的选择模型。若指定 null 值, 则使用空 selectionModel, 不允许选择
<code>getSelectionModel()</code>	返回选择的模型
<code>removeSelectionPath(TreePath path)</code>	从当前选择中移除指定路径标识的节点
<code>removeSelectionRow(int row)</code>	从当前选择移除索引 row 处的行
<code>clearSelection()</code>	清除该选择
<code>isSelectionEmpty()</code>	如果该选择当前为空, 则返回 true

表 11-4

其他方法及说明

<code>getDefaultTreeModel()</code>	创建并返回一个默认的 TreeModel
<code>createTreeModel(Object value)</code>	返回包装指定对象的 TreeModel。如果对象为 Object 数组, Hashtable 或 Vector, 则创建新的根节点, 将传入的每个对象作为子对象。否则创建新的根, 将指定的对象作为其值
<code>getModel()</code>	返回正在提供数据的 TreeModel
<code>setModel(TreeModel newModel)</code>	设置将提供数据的 TreeModel
<code>getRowCount()</code>	返回当前显示的行数
<code>hasBeenExpanded(TreePath path)</code>	如果已经展开路径标识的节点, 则返回 true

续表

<code>isExpanded(TreePath path)</code>	如果当前展开了路径标识的节点，则返回 <code>true</code>
<code>isExpanded(int row)</code>	如果当前展开了指定显示行处的节点，则返回 <code>true</code>
<code>isCollapsed(TreePath path)</code>	如果当前折叠了路径标识的值，则返回 <code>true</code> ，如果当前未显示路径中的任何值，则返回 <code>false</code>
<code>getPathForRow(int row)</code>	返回指定行的路径。如果 <code>row</code> 不可见，则返回 <code>null</code>
<code>getRowForPath(TreePath path)</code>	返回显示由指定路径标识的节点的行
<code>expandPath(TreePath path)</code>	确保指定路径标识的节点展开，并且可查看。如果路径中的最后一项是叶节点，则无效
<code>expandRow(int row)</code>	确保指定行中的节点展开，并且可查看
<code>collapsePath(TreePath path)</code>	确保指定路径标识的节点是折叠的，并且可查看
<code>collapseRow(int row)</code>	确保指定行中的节点是折叠的
<code>addTreeSelectionListener(TreeSelectionListener tsl)</code>	为 <code>TreeSelection</code> 事件添加侦听器
<code>removeTreeSelectionListener(TreeSelectionListener tsl)</code>	移除 <code>TreeSelection</code> 侦听器
<code>getTreeSelectionListeners()</code>	返回使用 <code>addTreeSelectionListener</code> 添加到此 <code>JTree</code> 的所有 <code>TreeSelectionListener</code> 的数组

另外 `JTree` 还可以使用继承父类的方法进行相应的操作，需要的读者可以查看相应的 API，这里不再赘述。

11.2.3 使用 `JTree` 的程序实例

本节开发了一个简单的使用 `JTree` 控件的程序，下面给出了该例的完整代码 (`DemoSimpleJTree.java`)。

```
package org.netbeans.swing.jtree;
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;
public class DemoSimpleJTree extends JFrame {
    JTree tree;
    //声明树模型
    DefaultTreeModel treemodel;
    //声明节点
    DefaultMutableTreeNode root, subroot, subroot2, leaf1, leaf2, leaf3, leaf4;
    public DemoSimpleJTree() {
        //创建根节点
        root=new DefaultMutableTreeNode("根节点");
        //创建子节点
        subroot=new DefaultMutableTreeNode("子节点 1");
        subroot2=new DefaultMutableTreeNode("子节点 2");
        //创建叶子节点
        leaf1=new DefaultMutableTreeNode("叶子节点 1");
        leaf2=new DefaultMutableTreeNode("叶子节点 2");
```

```
leaf3=new DefaultMutableTreeNode("叶子节点 3");
leaf4=new DefaultMutableTreeNode("叶子节点 4");
//使用给定的根节点创建树模型
treemodel =new DefaultTreeModel(root);
//使用指定的树模型创建树
tree=new JTree(treemodel);
//为根节点添加子节点
treemodel.insertNodeInto(subroot,root,0);
treemodel.insertNodeInto(subroot2,root,1);
//为子节点添加叶子节点
subroot.add(leaf1);
subroot.add(leaf2);
subroot2.add(leaf3);
subroot2.add(leaf4);
this.add(tree, BorderLayout.CENTER);
this.setTitle("JTree 简单应用");
this.setBounds(100,100,250,200);
this.setVisible(true);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String args[]) {
    new DemoSimpleJTree();
}
}
```

编译并运行程序，结果如图 11-2 所示。

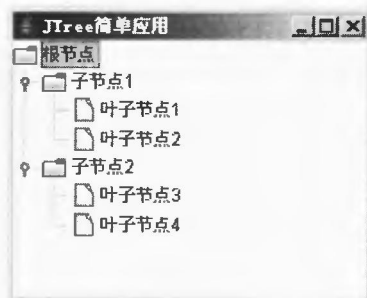


图 11-2 程序运行界面

本例中不仅使用了 JTree 控件，而且还用到了 DefaultMutableTreeNode（默认树节点）与 DefaultTreeModel（默认树模型），在本章后面的几节中将会对这些内容进行详细地介绍。

11.3 默认树模型

上一节给出的程序中使用了默认树模型（DefaultTreeModel），本节将会对默认树模型进行介绍，主要包括如下内容：

- DefaultTreeModel 类构造器介绍；

- DefaultTreeModel 的常用方法说明。

11.3.1 DefaultTreeModel 类构造器介绍

上节中已经提到 JTree 提供的只是数据的一个视图，并没有包含实际的数据。一般情况下它使用实现了 javax.swing.tree.TreeModel 接口的类作为其数据模型。

Swing 提供了一个实现了 TreeModel 接口的类 DefaultTreeModel（默认树模型），该类直接扩展了 java.lang.Object。一般情况下，都会使用该类作为 JTree 的数据模型。

DefaultTreeModel 类提供了如下两个构造器：

- DefaultTreeModel(TreeNode root)
- DefaultTreeModel(TreeNode root, boolean asksAllowsChildren)

1. 使用 DefaultTreeModel(TreeNode root)

使用 DefaultTreeModel(TreeNode root)构造器创建 DefaultTreeModel 对象必须提供一个根节点，下面代码介绍如何使用该构造器创建 DefaultTreeModel 对象。

```
TreeNode root=new DefaultMutableTreeNode("根节点");
DefaultTreeModel model=new DefaultTreeModel(root);
JTree myJTree=new JTree(model);
```

提示 **TreeNode** 是另外一个接口，**DefaultMutableTreeNode** 是实现了该接口的类，它具体的使用会在下节介绍。

2. 使用 DefaultTreeModel(TreeNode root, boolean asksAllowsChildren)

使用 DefaultTreeModel(TreeNode root, boolean asksAllowsChildren)构造器创建 DefaultTreeModel 对象，除了要提供一个根节点外，还要指定该节点是否可以有子节点。以下代码使用 DefaultTreeModel(TreeNode root, boolean asksAllowsChildren)构造器创建 DefaultTreeModel 对象。

```
DefaultTreeModel model=new DefaultTreeModel(root,false);
```

说明 以上代码创建了一个使用 root 作为根节点并不能有子节点的 DefaultTreeModel 对象。

11.3.2 DefaultTreeModel 的常用方法说明

创建 DefaultTreeModel 对象之后就可以使用 DefaultTreeModel 类提供的方法进行相应的操作。如可以使用 setAsksAllowsChildren(boolean newValue)方法设置某节点是否可以有子节点，代码如下：

```
model.setAsksAllowsChildren(true);
```

提示 根据传递给 setAsksAllowsChildren 方法的值，允许带有子节点的节点就可以得到非叶节点图标，而不允许带有子节点的节点则得到叶节点图标。

还可以使用 setRoot(TreeNode root)方法设置根节点，代码如下：

```
model.setRoot(myroot);
```


说明 以上代码将根节点设置为 myroot。

另外, 可以使用 DefaultTreeModel 类提供的其他方法进行相应的操作, 表 11-5 列出了其中一些常用的方法及说明。

表 11-5 DefaultTreeModel 的常用方法及说明

方 法	说 明
asksAllowsChildren()	告知如何确定叶节点
setRoot(TreeNode root)	将根设置为 root。root 为 null 表示树不显示任何内容, 这是合法的
getRoot()	返回树的根。当树没有节点时返回 null
getIndexOfChild(Object parent, Object child)	返回父节点中子节点的索引。如果父节点或子节点为 null, 则返回 -1
getChild(Object parent, int index)	返回父节点的子节点数组中索引 index 处的 parent 的子节点
getChildCount(Object parent)	返回 parent 的子节点数。如果该节点是叶节点或者它没有子节点, 则返回 0
isLeaf(Object node)	返回指定的节点是否为叶节点。执行测试的方法取决于 askAllowsChildren 设置
insertNodeInto(MutableTreeNode new Child, MutableTreeNode parent, int index)	对它进行调用, 以便在父节点的子节点中的 index 位置插入 newChild。然后, 通知 nodesWereInserted 创建适当的事件。这是添加子节点的首选方法, 因为它将创建适当的事件
removeNodeFromParent(MutableTreeNode node)	通知它从其父节点中移除节点。这将通知 nodesWereRemoved 创建适当事件。这是移除节点的首选方法
nodeChanged(TreeNode node)	改节点在树中的表示方式之后, 调用此方法
reload(TreeNode node)	如果已修改此模型依赖的 TreeNodes, 则调用此方法。该模型将通知其所有侦听器, 节点 node 下面的模型已更改
reload()	通知其所有侦听器, 节点 node 下面的模型已更改
nodesWereInserted(TreeNode node, int[] childIndices)	将某些 TreeNodes 插入节点之后, 调用此方法。子索引应是新元素的索引, 并且必须按升序排序
nodesWereRemoved(TreeNode node, int[] childIndices, Object[] removedChildren)	从节点移除一些 TreeNodes 之后, 调用此方法。childIndices 应是移除元素的索引, 并且必须按升序排序。removedChildren 应是移除的子对象的数组
nodesChanged(TreeNode node, int[] childIndices)	更改由 childIndices 标识的子对象在树中的表示方式之后, 调用此方法
nodeStructureChanged(TreeNode node)	如果完全更改了节点的子节点、子节点的子节点、依此类推, 则调用此方法。调用此方法将发布 treeStructureChanged 事件
getPathToRoot(TreeNode aNode)	向上构建节点的父节点一直到根节点 (包括根节点), 其中原始节点是返回数组中的最后一个元素。返回的数组长度给出了树中节点的深度
addTreeModelListener(TreeModelListener l)	为树更改后发布的 TreeModelEvent 添加侦听器
removeTreeModelListener(TreeModelListener l)	移除以前使用 addTreeModelListener() 添加的侦听器
getTreeModelListeners()	获取在此模型对象上注册的所有模型侦听器列表



提示 DefaultTreeModel 对于开发树状结构程序是非常重要的，应该熟练掌握它提供的方法的用法。

11.4 默认树节点

默认树节点 (DefaultMutableTreeNode) 是开发具有树状结构的程序时经常用到的一种控件，本节将会对默认树节点进行介绍，主要包括如下内容：

- DefaultMutableTreeNode 类构造器说明；
- DefaultMutableTreeNode 类的获取、添加及修改方法；
- DefaultMutableTreeNode 类的枚举方法。

11.4.1 DefaultMutableTreeNode 类构造器说明

读者可能注意到，DefaultTreeModel 类的使用依赖于 TreeNode 和 TreePath 对象。在树状结构中，TreeNode 代表存储在树中特定点的一个数据段，TreePath 代表若干个彼此直接关联的数据段集合。



说明 本节将为读者介绍 TreeNode 接口实现类 DefaultMutableTreeNode (默认树节点)，TreePath 的介绍将在下一节中给出。

DefaultMutableTreeNode 实现了 TreeNode 接口，主要用来创建一个节点，它提供了 3 个构造器。表 11-6 给出了这 3 个构造器及说明。

表 11-6 DefaultMutableTreeNode 构造器及说明

构造器	说 明
DefaultMutableTreeNode()	创建没有父节点和子节点的树节点，该树节点允许有子节点
DefaultMutableTreeNode(Object userObject)	创建没有父节点和子节点且允许有子节点的树节点，使用指定的用户对象对它进行初始化
DefaultMutableTreeNode(Object userObject, boolean allowsChildren)	创建没有父节点和子节点的树节点，使用指定的用户对象对它进行初始化，仅在 allowsChildren 为 true 时才允许有子节点

下面代码使用 DefaultMutableTreeNode() 构造器创建了一个名称为“新节点”的节点。

```
DefaultMutableTreeNode newNode=new DefaultMutableTreeNode("新节点");
```

可以看到，使用 DefaultMutableTreeNode() 构造器创建节点非常简单。



说明 对于使用另外两个构造器创建节点的情况，读者可以自己实验，这里就不给出具体的示例了。

11.4.2 DefaultMutableTreeNode 类的获取、添加及修改方法

创建 DefaultMutableTreeNode 对象之后，即可以使用 DefaultMutableTreeNode 类提供的获取、添加及修改方法获取节点、添加节点以及修改节点。

可以使用 getFirstChild 方法获取某节点的第一个子节点，代码如下：

```
myNode.getFirstChild();
```

可以使用 `add` 方法为某节点添加子节点，代码如下：

```
DefaultMutableTreeNode fatherNode=new DefaultMutableTreeNode("父节点");
DefaultMutableTreeNode sonNode=new DefaultMutableTreeNode("子节点");
fatherNode.add(sonNode);
```

 说明 `sonNode` 为 `fatherNode` 的子节点。

另外，还可以使用 `remove` 方法移除某个节点的子节点，代码如下：

```
fatherNode.remove(sonNode);
```

除了上面给出的几个方法，`DefaultMutableTreeNode` 还提供了大量的获取、添加及修改方法供开发人员使用，表 11-7 与表 11-8 列出了其中常用的方法及说明。

表 11-7 `DefaultMutableTreeNode` 的获取、添加及修改方法

方 法	说 明
<code>add(MutableTreeNode newChild)</code>	将指定节点 <code>newChild</code> 从父节点移除，并将其添加到此节点子节点数组的结尾，使其成为此节点的子节点
<code>getChildAfter(TreeNode aChild)</code>	返回此节点的子节点数组中 <code>aChild</code> 之后的子节点，它必须是此节点的子节点。如果 <code>aChild</code> 是最后一个子节点，则返回 <code>null</code>
<code>getChildBefore(TreeNode aChild)</code>	返回此节点的子节点数组中 <code>aChild</code> 之前的子节点，它必须是此节点的子节点。如果 <code>aChild</code> 是第一个子节点，则返回 <code>null</code>
<code>getLevel()</code>	返回此节点上的级数——从根到此节点的距离。如果此节点为根，则返回 0
<code>getDepth()</code>	返回以此节点为根的树的深度——从此节点到叶节点的最长距离。如果此节点没有子节点，则返回 0。此操作的开销远大于 <code>getLevel()</code> ，因为它必须有效地遍历以此节点为根的整棵树
<code>getFirstChild()</code>	返回此节点的第一个子节点。如果此节点没有子节点，则返回 <code>NoSuchElementException</code>
<code>getFirstLeaf()</code>	查找并返回此节点后代的第一个叶节点，即此节点或其第一个子节点的第一个叶节点。如果它是叶节点，则返回此节点
<code>getIndex(TreeNode aChild)</code>	返回此节点的子节点数组中指定子节点的索引。如果指定节点不是此节点的子节点，则返回 -1
<code>getLastChild()</code>	返回此节点的最后一个子节点。如果此节点没有子节点，则返回 <code>NoSuchElementException</code>
<code>getLastLeaf()</code>	查找并返回为此节点后代的最后一个叶节点，即此节点或其最后一个子节点的最后一个叶节点。如果它是叶节点，则返回此节点
<code>getLeafCount()</code>	返回此节点后代的叶节点总数。如果此节点是叶节点，则返回 1
<code>getNextLeaf()</code>	返回此节点后面的叶节点，如果此节点是树中的最后一个叶节点，则返回 <code>null</code>
<code>getNextNode()</code>	返回在此节点的树的前序遍历中此节点之后的节点。如果此节点是该遍历中的最后一个节点，则返回 <code>null</code>
<code>getNextSibling()</code>	返回父节点的子节点数组中此节点的下一个兄弟节点。如果此节点没有父节点或者是父节点的最后一个子节点，则返回 <code>null</code>
<code>getPath()</code>	返回从根到达此节点的路径。该路径中最后一个元素是此节点

续表

方 法	说 明
<code>getPreviousLeaf()</code>	返回此节点之前的叶节点, 如果此节点是树中的第一个叶节点, 则返回 <code>null</code>
<code>getPreviousNode()</code>	返回在此节点树的前序遍历中此节点之前的节点。如果此节点是遍历的第一个节点即树的根, 则返回 <code>null</code>
<code>getPreviousSibling()</code>	返回父节点的子节点数组中此节点的前一个兄弟节点。如果此节点没有父节点, 或者是父节点的第一个子节点, 则返回 <code>null</code>

表 11-8 `DefaultMutableTreeNode` 的获取、添加及修改方法续

方 法	说 明
<code>getRoot()</code>	返回包含此节点的树的根。根是所有节点的祖先, 其父节点为 <code>null</code>
<code>getSharedAncestor(DefaultMutableTreeNode aNode)</code>	返回此节点和 <code>aNode</code> 最近共同祖先。如果 <code>aNode</code> 是本节点的后代, 该节点就是要查找的共同的祖先。反之亦然
<code>getSiblingCount()</code>	返回此节点的兄弟节点数。节点是其本身的兄弟节点 (如果它没有父节点, 或者没有兄弟节点, 则此方法返回 1)
<code>getUserObjectPath()</code>	返回从根到达此节点的路径。如果路径中某些 <code>TreeNode</code> 的用户对象为 <code>null</code> , 则返回的路径将包含 <code>null</code>
<code>insert(MutableTreeNode newChild, int childIndex)</code>	从其现有父节点移除 <code>newChild</code> (如果它有父节点), 将子节点的父节点设置为此节点, 然后将该子节点添加到此节点位于索引 <code>childIndex</code> 处的子节点数组。 <code>newChild</code> 不能为 <code>null</code> , 也不能为此节点的祖先
<code>isLeaf()</code>	如果此节点没有子节点, 则返回 <code>true</code> 。要区分没有子节点的节点和不能拥有子节点的节点, 请将此方法与 <code>getAllowsChildren</code> 联合使用
<code>isNodeAncestor(TreeNode anotherNode)</code>	如果 <code>anotherNode</code> 是本节点的祖先, 返回 <code>true</code>
<code>isNodeChild(TreeNode anotherNode)</code>	如果 <code>anotherNode</code> 是本节点的子节点, 返回 <code>true</code>
<code>isNodeDescendant(DefaultMutableTreeNode anotherNode)</code>	如果 <code>anotherNode</code> 是本节点的后代, 返回 <code>true</code>
<code>isNodeRelated(DefaultMutableTreeNode aNode)</code>	当且仅当 <code>aNode</code> 位于与此节点相同的树中时, 返回 <code>true</code> 。如果 <code>aNode</code> 为 <code>null</code> , 则返回 <code>false</code>
<code>isNodeSibling(TreeNode anotherNode)</code>	如果 <code>anotherNode</code> 是此节点的兄弟节点 (与此节点共享同一父节点), 则返回 <code>true</code>
<code>isRoot()</code>	如果此节点是树的根, 则返回 <code>true</code>
<code>remove(int childIndex)</code>	从此节点的子节点中移除指定索引处的子节点, 并将该节点的父节点设置为 <code>null</code>
<code>remove(MutableTreeNode aChild)</code>	通过向其提供一个 <code>null</code> 父节点, 从此节点的子节点数组移除 <code>aChild</code>
<code>removeAllChildren()</code>	通过将其父节点设置为 <code>null</code> , 移除此节点的所有子节点。如果此节点没有子节点, 则此方法不执行任何操作
<code>removeFromParent()</code>	通过向此节点提供一个 <code>null</code> 父节点, 从树中移除此节点为根的子树



说明 表 11-7 与表 11-8 列出了 `DefaultMutableTreeNode` 类的所有获取、添加及修改方法, 如果读者需要进行相应的操作, 可以从这两个表中查找合适的方法。

11.4.3 DefaultMutableTreeNode 类的枚举方法


除了上一节中提到的获取、添加及修改方法，DefaultMutableTreeNode 类还提供了一些枚举方法供开发人员使用，表 11-9 列出了这些枚举方法及说明。

表 11-9 DefaultMutableTreeNode 的枚举方法及说明

方 法	说 明
breadthFirstEnumeration()	创建并返回一个枚举，该枚举按广度优先的顺序遍历以此节点为根的子树。由枚举的 nextElement 方法返回的第一个节点是此节点
depthFirstEnumeration()	创建并返回一个枚举，该枚举按深度优先的顺序遍历以此节点为根的子树。由枚举的 nextElement 方法返回的第一个节点是最左边的叶节点。这与后序遍历相同
postorderEnumeration()	创建并返回一个枚举，该枚举后序遍历以此节点为根的子树。由枚举的 nextElement 方法返回的第一个节点是最左边的叶节点。这与深度优先遍历相同
preorderEnumeration()	创建并返回一个枚举，该枚举前序遍历以此节点为根的子树。由枚举的 nextElement 方法返回的第一个节点是此节点
children()	创建并返回此节点的子节点的正向枚举
PathFromAncestorEnumeration(TreeNode ancestor)	创建并返回沿着从 ancestor 到此节点的路径的一个枚举。枚举的 nextElement 方法首先返回 ancestor，然后返回作为此节点祖先的 ancestor 的子节点，依次类推，最后返回此节点

可以使用 breadthFirstEnumeration 方法获取所有以 root 为父节点的节点，代码如下：

```
//root 为根节点
Enumeration e=root.breadthFirstEnumeration();
//遍历方法
while(e.hasMoreElements()){
    DefaultMutableTreeNode node=(DefaultMutableTreeNode)e.nextElement();
    //打印根节点的内容
    System.out.println(node.getUserObject());
}
```

 **说明** 每个枚举方法，都有与之功能相同的获取方法，不过枚举方法比相应的获取方法效率要高得多。因此在实际项目开发中应该使用枚举方法。

11.5 树路径

树路径（TreePath）是开发具有树状结构的程序时经常使用的一种控件，本节将会对 TreePath 类进行介绍，主要包括如下内容：

- TreePath 类的构造器；
- TreePath 类的常用方法说明。

11.5.1 TreePath 类的构造器

TreePath（树路径）类掌管着一系列 Object 类型的引用，许多 JTree 类对象的方法都可以

返回 `TreePath` 对象。

提示 `TreePath` 对象提供的只是路径的视图。如果要改变路径的结构,则需要改变模型而不是改变 `TreePath` 对象。

`TreePath` 类提供了 5 个构造器,表 11-10 列出了这 5 个构造器及其说明。

表 11-10 `TreePath` 类构造器及说明

构造器	说明
<code>TreePath(Object[] path)</code>	从 <code>Objects</code> 的数组构造路径,并根据树的数据模型的返回情况惟一标识树的根到指定节点的路径
<code>TreePath(Object singlePath)</code>	构造仅包含单个元素的 <code>TreePath</code> 。这通常用于构造 <code>TreeModel</code> 的根的 <code>TreePath</code>
<code>TreePath(TreePath parent, Object lastElement)</code>	构造一个新的 <code>TreePath</code> ,它是以 <code>lastElement</code> 结束的 <code>parent</code> 标识的路径
<code>TreePath(Object[] path,int length)</code>	构造一个具有长度为 <code>length</code> 的标识路径组件的新 <code>TreePath</code>
<code>TreePath()</code>	主要为以不同方式表示路径的子类提供。如果子类使用此构造方法,则它还应重写 <code>getPath</code> 、 <code>getPathCount</code> 和 <code>getPathComponent</code> 方法,并可能重写 <code>equals</code> 方法

说明 虽然 `TreePath` 类提供了以上几个构造器创建 `TreePath` 对象,但在实际项目开发中,并不会直接创建 `TreePath` 对象。一般情况下都是使用 `JTree` 或默认树模型的相关方法获取 `TreePath` 对象。

11.5.2 `TreePath` 类的常用方法说明

当获取了 `TreePath` 对象之后,就可以使用 `TreePath` 类的相应方法进行操作。例如,使用 `getLastPathComponent` 方法获取此路径的最后一个组件,代码如下:

```
selectionPath.getLastPathComponent();
```

使用 `getParentPath` 方法获取包含除最后一个路径组件之外的所有元素路径,代码如下:

```
selectionPath.getParentPath();
```

虽然 `TreePath` 类比较简单,但也包含了一些比较路径以及处理路径的简单方法,表 11-11 列出了其中一些常用的方法及说明。

表 11-11 `TreePath` 类的常用方法及说明

方法	说明
<code>getPath()</code>	返回有序的 <code>Objects</code> 数组,它包含此 <code>TreePath</code> 的组件。第一个元素(index 为 0)是根
<code>getLastPathComponent()</code>	返回此路径的最后一个组件。对于 <code>DefaultTreeModel</code> 返回的路径,它将返回一个 <code>TreeNode</code> 实例
<code>getPathCount()</code>	返回路径中的元素数
<code>getPathComponent(int element)</code>	返回指定索引位置的路径组件
<code>equals(Object o)</code>	通过检查路径中每个元素的相等性,测试两个 <code>TreePaths</code> 的相等性。如果两个路径等长并且包含相同的元素,则认为它们相等

续表

方 法	说 明
isDescendant(TreePath aTreePath)	如果 aTreePath 为此 TreePath 的后代, 则返回 true
pathByAddingChild(Object child)	返回包含此对象的所有元素加上 child 的新路径。child 将是新创建的 TreePath 的最后一个元素
getParentPath()	返回包含除最后一个路径组件之外的此对象所有元素的路径



说明

TreePath 类的使用方法相对简单, 使用上述方法基本上可以完成与 TreePath 有关的所有操作。对于上述方法的详细说明, 读者可以查阅相关的 API。

11.6 树的事件

当对树进行相关操作时, 会触发相应的事件。本节将会对与树操作有关的事件进行介绍, 主要包括如下内容:

- 选择事件;
- 扩展事件;
- 模型结构变化事件。

11.6.1 选择事件

当用户 (或程序) 更改树的选中节点时, 就会触发选择事件 (TreeSelectionEvent)。TreeSelectionEvent 封装了一些与选择有关的信息, 表 11-12 列出了 TreeSelectionEvent 类提供的获取这些信息的方法及其说明。

表 11-12

TreeSelectionEvent 类的常用方法及说明

方 法	说 明
getPaths()	返回已经添加到该选择中或已从中移除的路径
getPath()	返回第一个路径元素
isAddedPath()	如果已经将第一个路径元素添加到选择中, 则返回 true。返回 false 值意味着第一条路径已从选择中移除
isAddedPath(TreePath path)	如果 path 标识的路径被添加到选择中, 则返回 true。返回 false 值意味着该路径过去在该选择中, 但现在已不在该选择中
isAddedPath(int index)	如果 index 标识的路径被添加到选择中, 则返回 true。返回 false 值意味着该路径过去在该选择中, 但现在已不在该选择中
getOldLeadSelectionPath()	返回以前的前导路径
getNewLeadSelectionPath()	返回当前前导路径

TreeSelectionEvent 事件使用实现了 TreeSelectionListener 接口的类来处理。该接口中只包含 valueChanged 一个方法, 对 TreeSelectionEvent 事件的处理代码位于该方法的方法体内。

11.6.2 扩展事件

当用户 (或程序) 展开或折叠某一节点时会产生扩展事件 (TreeExpansionEvent)。TreeExpansionEvent 类只提供了 getPath 方法返回到达被折叠 (或展开) 节点的路径。

`TreeExpansionEvent` 事件通过使用实现了 `TreeExpansionListener` 接口的监听器类对象来处理, 该接口中包含 `treeExpanded` 与 `treeCollapsed` 两个方法。当展开节点时, `treeExpanded` 方法会被调用, 而当折叠节点时, `treeCollapsed` 方法会被调用。

11.6.3 模型结构变化事件

当用户更改模型时, 会触发模型结构变化事件 (`TreeModelEvent`), 该类对象封装了描述树模型变化的信息, 表 11-13 列出了获取这些信息的方法及其说明。

表 11-13 `TreeModelEvent` 的常用方法及说明

方 法	说 明
<code>getTreePath()</code>	返回已更改节点的父节点。可使用此方法和 <code>getChildIndices</code> 来获得受影响节点的列表
<code>getPath()</code>	从此事件包装的 <code>TreePath</code> 实例中获得对象数组的便捷方法
<code>getChildren()</code>	返回位于 <code>getChildIndices</code> 指定位置处的、作为 <code>getPath</code> 所标识节点的子节点的对象
<code>getChildIndices()</code>	返回子索引的值

`TreeModelEvent` 事件通过使用实现了 `TreeModelListener` 接口的监听器类对象来处理。`TreeModelListener` 接口提供了 4 个方法用来处理相应的动作。表 11-14 给出了这些方法及说明。

表 11-14 `TreeModelListener` 的事件处理方法及说明

方 法	说 明
<code>treeNodesChanged(TreeModelEvent e)</code>	在已经以某种方式更改节点后调用
<code>treeNodesInserted(TreeModelEvent e)</code>	在已将节点插入树中以后调用
<code>treeNodesRemoved(TreeModelEvent e)</code>	在已从树中移除节点后调用
<code>treeStructureChanged(TreeModelEvent e)</code>	在树结构中, 从某个给定节点开始向下的节点发生更改之后调用该方法



说明

本节并没有给出具体的示例来介绍这些事件的使用, 在后面的章节中给出了一个综合使用这 3 种事件的示例。

11.7 树单元绘制器

在应用程序中, 可能常常需要自定义树控件中节点的表示方法, 如为非叶节点和叶节点选择不同的图标, 这些都要用到树单元绘制器。本节对与树单元绘制器有关的内容进行介绍, 主要包括如下内容:

- 默认树单元绘制器;
- 自定义树单元绘制器。

11.7.1 默认树单元绘制器

前面已经提到, 如果改变树控件中节点的表示方法, 要用到树单元绘制器。默认情况下, `JTree` 类使用 `DefaultTreeCellRenderer` 对象来控制对每个节点的绘制。该类扩展了 `JLabel`, 包

含节点图标和节点文字。

`DefaultTreeCellRenderer` 类只提供了一个空构造器, 下列代码使用该构造器代码创建了一个 `DefaultTreeCellRenderer` 对象。

```
DefaultTreeCellRenderer render=new DefaultTreeCellRenderer();
```

创建 `DefaultTreeCellRenderer` 对象以后, 即可以使用 `DefaultTreeCellRenderer` 类提供的方法对节点绘制进行相应的操作, 主要用法如下:

● 设置叶节点图标

```
render.setLeafIcon(new ImageIcon("leaf.gif"));
```

● 设置非叶节点折叠图标

```
render.setClosedIcon(new ImageIcon("closed.gif"));
```

● 设置非叶节点展开图标

```
render.setOpenIcon(new ImageIcon("open.gif"));
```

● 将 `render` 设置为 `tree` 的单元绘制器

```
tree.setCellRenderer(render);
```

说明 `render` 为默认绘制器对象句柄, `tree` 为树对象句柄。

`DefaultTreeCellRenderer` 类还提供了很多方法进行相应的操作, 表 11-15 列出了其中一些常用的方法及说明。

表 11-15 `DefaultTreeCellRenderer` 常用方法及说明

方 法	说 明
<code>getDefaultOpenIcon()</code>	返回展开节点时的默认图标
<code>getDefaultClosedIcon()</code>	返回折叠节点时的默认图标
<code>getDefaultLeafIcon()</code>	返回叶子节点的默认图标
<code>setOpenIcon(Icon newIcon)</code>	设置用于表示扩展的非叶节点的图标
<code>getOpenIcon()</code>	返回用于表示扩展的非叶节点的图标
<code>setClosedIcon(Icon newIcon)</code>	设置用于表示没有扩展的非叶节点的图标
<code>getClosedIcon()</code>	返回用于表示没有扩展的非叶节点的图标
<code>setLeafIcon(Icon newIcon)</code>	设置用于表示叶节点的图标
<code>getLeafIcon()</code>	返回用于表示叶节点的图标
<code>setTextSelectionColor(Color newColor)</code>	设置选定节点时绘制文本所使用的颜色
<code>getTextSelectionColor()</code>	返回选定节点时绘制文本所使用的颜色
<code>setTextNonSelectionColor(Color newColor)</code>	设置未选定节点时绘制文本所使用的颜色
<code>getTextNonSelectionColor()</code>	返回未选定节点时绘制文本所使用的颜色
<code>setBackgroundSelectionColor(Color newColor)</code>	设置在选定节点的情况下背景所使用的颜色
<code>getBackgroundSelectionColor()</code>	返回在选定节点的情况下背景所使用的颜色
<code>setBackgroundNonSelectionColor(Color newColor)</code>	设置非选定节点的背景色

续表

方 法	说 明
<code>getBackgroundNonSelectionColor()</code>	返回非选定节点的背景色
<code>setBorderSelectionColor(Color newColor)</code>	设置边框的颜色
<code>getBorderSelectionColor()</code>	返回绘制边框时所使用的颜色
<code>setFont(Font font)</code>	设置字体的颜色
<code>getFont()</code>	获取此组件的字体



说明

可以使用表 11-15 中所列出的方法更改 `DefaultTreeCellRenderer` 使用的图标、字体和背景颜色。这些设置将被应用于树中所有的节点。

11.7.2 自定义单元绘制器

若要单独设置各个节点的外观，则应该自定义单元绘制器并将其设置为树的单元绘制器。自定义单元绘制器一般情况下都会扩展 `DefaultTreeCellRenderer` 类，并按照如下步骤覆盖其中的 `getTreeCellRendererComponent` 方法。

- (1) 调用超类的方法。
- (2) 定制标签的属性。
- (3) 返回 `this`。

下面给出了一个自定义单元绘制器的示例代码：

```
class myTreeCellRenderer extends DefaultTreeCellRenderer{
    public Component getTreeCellRendererComponent(JTree tree, Object value,
        boolean selected, boolean expanded, boolean leaf, int row, boolean
        hasFocus) {
        super.getTreeCellRendererComponent(tree, value, selected, expanded, leaf,
            row, hasFocus);
        /*
         * 此处添加设置标签属性的代码。如设置背景颜色，设置字体大小，设置图标等
         */
        /*
         * 返回该类的一个实例，DefaultTreeCellRenderer 扩展了 JLabel，可以认为该方法返回
         * 的是 JLabel 的子类对象
         */
        return this;
    }
}
```



提示

`getTreeCellRendererComponent` 方法的 `value` 参数是一个节点对象，而不是一个用户对象。另外 `DefaultTreeCellRenderer` 将同样的标签对象用于所有的节点。

自定义单元绘制器以后，可以使用 `JTree` 类的 `setCellRenderer` 方法将其设置为树的单元绘制器，代码如下：

```
tree.setCellRenderer(new myTreeCellRenderer());
```

通过使用自定义单元绘制器，可以使程序的界面更加具有用户特色。在开发具有树状结构的程序时，如果需要可以使用自定义单元绘制器。

11.8 在 NetBeans 中开发树状结构的程序实例

前面几节介绍了与树有关的一些知识，本节将综合使用这些知识开发一个具有树状结构的程序，主要包括如下内容：

- 项目概述；
- 界面设计；
- 初始化操作及处理节点的选择，展开及更改事件；
- 增加与删除节点；
- 更改节点图标；
- 项目总结。

11.8.1 项目概述

本节开发了一个具有树状结构的程序，程序运行界面如图 11-3 所示，主要有如下功能：


- 在程序中选中节点后，在窗口下端的信息提示面板中会出现“你选择了 XXX 节点”的信息，展开与折叠节点也会出现相应的提示信息。
- 选择右键菜单中的“添加节点”与“删除节点”，可以为选中的节点添加子节点或者删除选中的节点。
- 如果想删除根节点，则会弹出“XXX 是根节点，你不能删除！”的信息提示对话框。
- 选择右键菜单中的“更改展开节点图标”、“更改折叠节点图标”或“更改叶节点图标”可以为节点设置相应的图标。



图 11-3 程序运行界面

11.8.2 界面设计

JTree 是典型的 Swing MVC 控件，它使用模型封装数据，因此本项目的界面设计相对比较容易。启动 NetBeans，按如下步骤完成该例中界面的设计。

- (1) 创建名称为 tree 的项目并将其主类的名称设置为 org.netbeans.swing.tree.Demo。
- (2) 向 tree 中添加一个通过 JFrame Form 模版创建的类，名称为 DemoJTree。
- (3) 向 DemoJTree 中添加 JTree，将其名称修改为 jTreeCountry。
- (4) 打开 jTreeCountry 的属性设置窗口，单击 model 属性右侧的  按钮，弹出如图 11-4 所示窗口。
- (5) 在窗口中设置 model 属性。选中“User Code”单选按钮，将下列代码添加到 User Code 文本框中。

```
this.initialTreeModel();
```

提示 initialTreeModel 是 DemoJTree 中一个自定义的方法，其返回 DefaultTreeModel，用来初始化树的模型，后面将会讲解该方法。

(6) 向 DemoJTree 中添加一个 JLabel，将其名称修改为 jLabelMessage 并将其 text 属性值设置为“您还没有做出任何动作”。

(7) 此时，界面设计就已经完成了。GUI 设计器中内容如图 11-5 所示。

图中的各个节点是添加 JTree 时自动生成的。为 JTree 设置新的模型以后，其各个节点会发生变化，甚至不再存在，但 GUI 设计器中内容不变。在 NetBeans 中使用树控件要以实际运行界面为准。

(8) 将下列代码添加到 Demo 类的 public static void main(string args[]) 方法的方法体内。

```
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoJTree().setVisible(true);
    }
});
```

编译并运行项目，结果如图 11-6 所示。

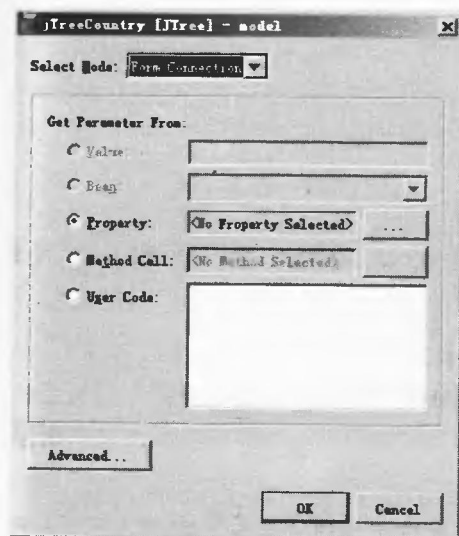


图 11-4 设置 model 属性



图 11-5 GUI 设计器中内容

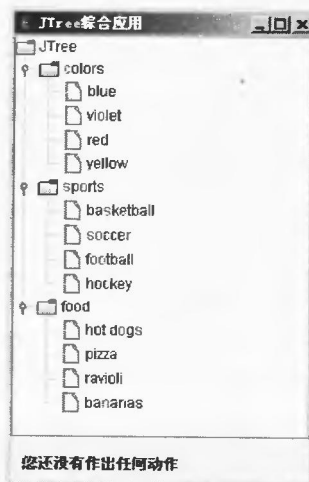


图 11-6 程序运行界面

此时程序还没有具体的功能，功能代码的开发在后面的小节中介绍。

11.8.3 初始化操作及处理节点的选择，展开及更改事件

(1) 向 DemoJTree 中添加名为 private DefaultTreeModel initialTreeModel 的方法，并将下列代码添加到该方法的方法体内。

```
DefaultMutableTreeNode root=new DefaultMutableTreeNode("中国");//创建根节点
DefaultMutableTreeNode subRoot1=new DefaultMutableTreeNode("北京");//创建子节点
```

```

DefaultMutableTreeNode subRoot2=new DefaultMutableTreeNode("上海");
DefaultMutableTreeNode subRoot3=new DefaultMutableTreeNode("河北");
DefaultMutableTreeNode subRoot4=new DefaultMutableTreeNode("河南");
DefaultMutableTreeNode subRoot5=new DefaultMutableTreeNode("广东");
DefaultMutableTreeNode subRoot6=new DefaultMutableTreeNode("辽宁");
DefaultMutableTreeNode leaf1=new DefaultMutableTreeNode("唐山");//创建叶子节点
DefaultMutableTreeNode leaf2=new DefaultMutableTreeNode("廊坊");
DefaultMutableTreeNode leaf3=new DefaultMutableTreeNode("郑州");
DefaultMutableTreeNode leaf4=new DefaultMutableTreeNode("洛阳");
DefaultMutableTreeNode leaf5=new DefaultMutableTreeNode("广州");
DefaultMutableTreeNode leaf6=new DefaultMutableTreeNode("佛山");
DefaultMutableTreeNode leaf7=new DefaultMutableTreeNode("大连");
DefaultMutableTreeNode leaf8=new DefaultMutableTreeNode("沈阳");
root.add(subRoot1);//向根节点中添加子节点
root.add(subRoot2);
root.add(subRoot3);
root.add(subRoot4);
root.add(subRoot5);
root.add(subRoot6);
subRoot3.add(leaf1);//向子节点中添加叶子节点
subRoot3.add(leaf2);
subRoot4.add(leaf3);
subRoot4.add(leaf4);
subRoot5.add(leaf5);
subRoot5.add(leaf6);
subRoot6.add(leaf7);
subRoot6.add(leaf8);
DefaultTreeModel myDefaultTreeModel=new DefaultTreeModel(root);//使用根
节点创建模型
//向 myDefaultTreeModel 添加 TreeModelEvent 事件监听器
//DealAction 为实现了 TreeModelListener 接口的类
myDefaultTreeModel.addTreeModelListener(new DealAction());
return myDefaultTreeModel;

```

本段代码中主要包含如下内容。

- ① 创建了多个 DefaultMutableTreeNode 对象并将其添加到相应的父节点。
- ② 创建了 DefaultTreeModel 对象，将其作为 JTreeCountry 的模型。
- ③ 为 DefaultTreeModel 对象添加了 TreeModelEvent 事件监听器。

(2) 向 DemoJTree 中添加名称为 actionNode，类型为 DefaultMutableTreeNode，访问限制修饰符为 private，初始值为 null 的成员变量。

(3) 向 DemoJTree 中添加名称为 DealAction 的内部类。该类实现了 TreeSelectionListener 接口、TreeExpansionListener 接口和 TreeModelListener 接口，并将下列代码添加到该内部类的类体内：

```

//处理节点的选择事件
public void valueChanged(TreeSelectionEvent event1) {

```



```

        TreePath tp=event1.getNewLeadSelectionPath();//获取新选中节点的路径
        if(tp!=null){
            DefaultMutable tempNode=(DefaultMutableTreeNode)tp.get
            LastPathComponent();
            DemoJTree.this.jLabelMessage.setText("你选择了:"+ tempNode);
            DemoJTree.this.actionNode= tempNode;// actionNode 用来记录当
            前选中节点
        }
    }
    //处理节点的展开事件
    public void treeExpanded(TreeExpansionEvent event2) {
        TreePath tp=event2.getPath();//获取展开节点的路径
        DemoJTree.this.jLabelMessage.setText("你展开了:"+tp.getLastPath
        Component());
    }
    //处理节点折叠事件
    public void treeCollapsed(TreeExpansionEvent event3) {
        TreePath tp=event3.getPath();//获取折叠节点的路径
        DemoJTree.this.jLabelMessage.setText("你将"+tp.getLastPath-
        Component()+"节点折叠");
    }
    //处理节点内容改变事件
    public void treeNodesChanged(TreeModelEvent e) {
        DemoJTree.this.jLabelMessage.setText("你更改了节点的内容");
    }
    //以下三个方法为 TreeModelListener 接口中方法, 这里使用空实现
    public void treeNodesInserted(TreeModelEvent e) { }
    public void treeNodesRemoved(TreeModelEvent e) { }
    public void treeStructureChanged(TreeModelEvent e) { }

```

(4) 向 DemoJTree 中添加签名为 `private void initialTree` 的方法。该方法对 `JTreeCountry` 对象进行了简单的设置, 并为其添加相应事件的监听器, 然后将下列代码添加到该方法的方法体内:

```

        this.jTreeCountry.setEditable(true);//将 JTreeCountry 的节点设置为可编辑,
        默认为不可编辑
        this.jTreeCountry.setShowsRootHandles(true);//设置树的根节点具有“门把手图标”
        DealAction dealAction=new DealAction();//创建监听器对象
        this.jTreeCountry.addTreeSelectionListener(dealAction);//注册选择监听器对象
        this.jTreeCountry.addTreeExpansionListener(dealAction); //注册展开监听器对象

```

(5) 将下列代码添加到 DemoJTree 类的构造器内:

```

        this.initialTree();

```

(6) 编译并运行项目后, 结果与图 11-3 所示一致。选中“北京”节点, `JLabelMessage` 会显示“你选择了: 北京”, 如图 11-7 所示。

(7) 展开“河北”节点, `JLabelMessage` 会显示“你展开了: 河北节点”, 如图 11-8 所示。

(8) 折叠“河北”节点，jLabelMessage 会显示“你将河北节点折叠”，如图 11-9 所示。

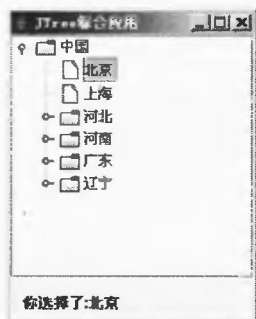


图 11-7 选中“北京”节点

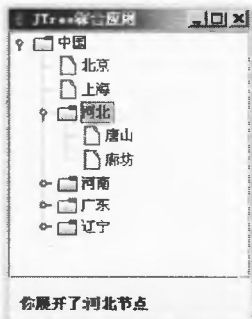


图 11-8 展开“河北”节点

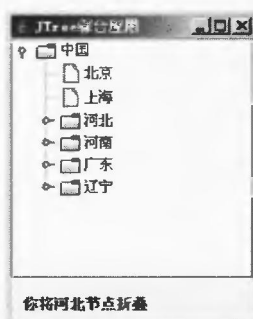


图 11-9 折叠“河北”节点

(9) 通过在相应节点上连续单击 3 次，节点标签处于编辑状态，如图 11-10 所示。更改节点的内容，将其修改为“上海之家”，并按下回车键，此时界面如图 11-11 所示。

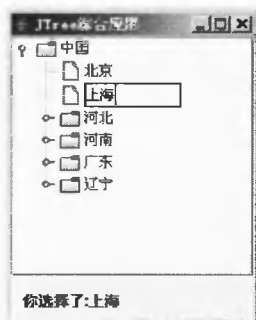


图 11-10 编辑节点标签内容

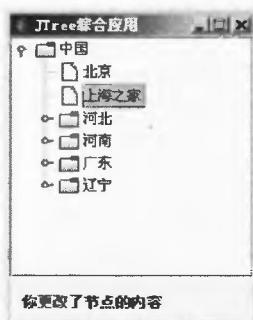


图 11-11 更改节点标签的内容

对不同的节点进行相应的操作，jLabelMessage 会显示相应的动作信息。有兴趣的读者可以做实验，这里不再详细介绍。

11.8.4 增加与删除节点

上一节对项目进行了初始化操作并处理了节点的选择、展开、折叠与更改节点内容等事件。本节将主要讲述如何添加和删除节点，按照如下步骤完成该例。

(1) 向 DemoJTree 中添加名称为 jPopupMenuNode 的 JPopupMenu。

(2) 依次向 jPopupMenuNode 中添加名称为 jMenuItemAddNode 和 jMenuItemDeleteNode 的 JMenuItem，分别将其 text 属性的值设置为“添加节点”与“删除节点”。

(3) 向 DemoJTree 中添加名称为 PopupAdapter 的扩展了 MouseAdapter 的内部类。将下列代码添加到该内部类的类体内：

```
JPopupMenu popup;
PopupAdapter(JPopupMenu popupMenu) {
    popup = popupMenu;
}
public void mousePressed(MouseEvent e) {
    showPopupMenu(e);
}
```

```

    }
    public void mouseReleased(MouseEvent e) {
        showPopupMenu(e);
    }
    private void showPopupMenu(MouseEvent e) {
        if (e.isPopupTrigger()) {
            popup.show(e.getComponent(), e.getX(), e.getY());
        }
    }
}

```



说明 此内部类对象用来作为鼠标事件监听器,如果对应的鼠标事件为弹出式菜单触发器则弹出菜单。

(4) 为 `jMenuItemAddNode` 与 `jMenuItemDeleteNode` 添加 `ActionEvent` 事件处理方法,并按如下所述的方法向相应的方法体内添加代码。

● 向 `jMenuItemAddNodeActionPerformed` 方法中添加如下代码:

```

String nodeName=JOptionPane.showInputDialog(this,"请输入要添加的节点的名称",
        "请输入: ",JOptionPane.INFORMATION_MESSAGE);
if (nodeName!=null&&!nodeName.equals("")) {
    if (this.actionNode!=null) { // actionNode 表示当前被选中的节点
        DefaultMutableTreeNode newNode=new DefaultMutableTreeNode
(nodeName);
        actionNode.add(newNode); // 为当前选中节点添加子节点
        // 通知所有注册的监听器模型已改并且重新加载模型
        ((DefaultTreeModel) this.jTreeCountry.getModel()).reload
(actionNode);
        this.jTreeCountry.setExpandsSelectedPaths(true); // 更改选择
        // 使其父节点可见
        this.jLabelMessage.setText(newNode + "被添加");
    }
}

```

● 向 `jMenuItemDeleteNode` 方法中添加如下代码:

```

int confirm=JOptionPane.showConfirmDialog(this,"你确定要删除该节点吗?",
        "提示: ",JOptionPane.YES_NO_OPTION,JOptionPane.QUESTION_MESSAGE);
if (confirm==JOptionPane.YES_OPTION) {
    // 获取当前选中节点的父节点
    DefaultMutableTreeNode parent=(DefaultMutableTreeNode) actionNode.
getParent();
    if (actionNode!=null) {
        if (parent==null) { // 如果当前选择节点是根节点, 则不能删除
            JOptionPane.showMessageDialog(this,actionNode+"是根节点,
            不能删除!", "警告",JOptionPane.WARNING_MESSAGE);
        } else {
            this.jLabelMessage.setText(actionNode+"被删除了!");
            parent.remove(this.actionNode); // 从父节点中移除当前选中节点
        }
    }
}

```

```

        this.actionNode=null;//将当前选中节点设置为 null
        ((DefaultTreeModel)this.jTreeCountry.getModel()).
reload(parent);
    }
}
}

```

(5) 向 `initialTree` 方法的方法体内添加如下代码:

```
this.jTreeCountry.addMouseListener(new PopupAdapter(this.jPopupMenuNode));
```

(6) 编译并运行程序, 结果如图 11-3 所示。右键单击“广东”节点, 结果如图 11-12 所示。选择“添加节点”选项, 将弹出输入窗口, 如图 11-13 所示。

(7) 在图 11-13 所示窗口中输入“深圳”, 单击“确定”按钮后, 展开“广东”节点, 确实可以看到在其下添加了“深圳”节点, 如图 11-14 所示。

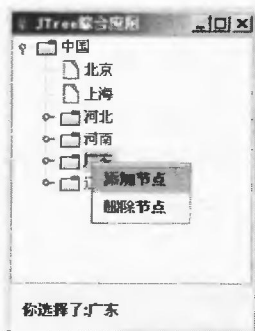


图 11-12 广东节点的右键菜单

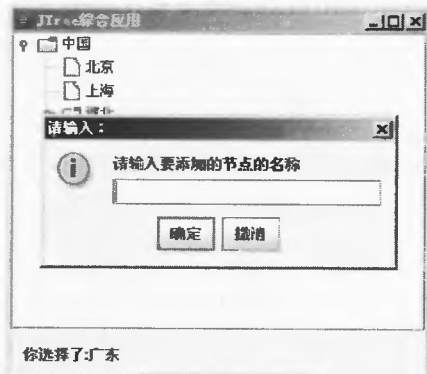


图 11-13 添加节点



图 11-14 添加“深圳”

(8) 单击鼠标右键, 在弹出的菜单中选择“删除节点”选项, 可以删除选中的节点。如果选中节点是根节点, 则程序会提示不能删除该节点; 如果选中的节点含有子节点, 则连同子节点一起删除。

11.8.5 更改节点图标

上一节介绍了如何添加、删除树中节点, 本节将为读者介绍如何修改节点所使用的图标, 按照如下步骤完成该例。

(1) 向 `jPopupMenuNode` 中添加 3 个 `JMenuItem` 并按照表 11-16 所示修改其属性。

表 11-16

属性与值对照表

控件名称	text 属性值
<code>jMenuItemChangeOpenIcon</code>	更改展开节点图标
<code>jMenuItemChangeCloseIcon</code>	更改折叠节点图标
<code>jMenuItemLeafIcon</code>	更改叶节点图标

(2) 向 `DemoJTree` 中添加名称为 `fileChooser`、类型为 `JFileChooser`、访问限制修饰符为

private、初始值为 new JFileChooser 的成员变量。

(3) 向 DemoJTree 中添加名为 private ImageIcon chooserIcon 的方法。该方法主要用来从用户的文件系统中加载 ImageIcon 对象并返回该对象，然后将下列代码添加到该方法的方法体内。

```
ImageIcon tempIcon=null;
fileChooser.showOpenDialog(this);
File iconFile=fileChooser.getSelectedFile();
if(iconFile!=null){
    ImageIcon newIcon=new ImageIcon(iconFile.getPath());
    tempIcon=newIcon;
}
return tempIcon;
```

(4) 分别为 jMenuItemChangeOpenIcon、jMenuItemChangeCloseIcon、jMenuItemLeafIcon 添加 ActionEvent 事件处理方法。



(5) 向 DemoJTree 中添加名为 private void dealTheChangeIconEvent(ActionEvent evt) 的方法，并将下列代码添加到该方法中：

```
ImageIcon newIcon=this.chooserIcon();//获取 ImageIcon 对象
DefaultTreeCellRenderer renderer
    =(DefaultTreeCellRenderer)this.jTreeCountry.getCellRenderer(); //获取
jTreeCountry 的单元绘制器
if(evt.getSource()==jMenuItemChangeOpenIcon){
    renderer.setOpenIcon(newIcon);//设置展开标签的图标
}
else if(evt.getSource()==jMenuItemChangeCloseIcon){
    renderer.setClosedIcon(newIcon); //设置折叠标签的图标
}
else if(evt.getSource()==jMenuItemLeafIcon){
    renderer.setLeafIcon(newIcon);//设置叶子节点的图标
}
this.jTreeCountry.setCellRenderer(renderer);//将 renderer 设置为 jTreeCountry 的单
元绘制器
((DefaultTreeModel)this.jTreeCountry.getModel()).reload();
```

(6) 分别将下列代码添加到 jMenuItemChangeOpenIcon、jMenuItemChangeCloseIcon 和 jMenuItemLeafIcon 的 ActionEvent 事件处理方法中：

```
this.dealTheChangeIconEvent(evt);
```

(7) 编译并运行程序，结果如图 11-3 所示。右键单击“大连”节点，选择“更改叶节点图标”选项，将弹出选择图标的窗口，如图 11-15 所示。选择好要使用的图标以后，单击“打开”按钮，此时结果如图 11-16 所示。

从图 11-16 中可以看出，叶节点的图标已经不是系统默认的，而是用户自定义的了。

通过选择右键菜单中的“更改展开节点”图标和“更改折叠节点”图标还可以设置展开节点和折叠节点的图标，这里不再赘述。



图 11-15 选择使用的图标

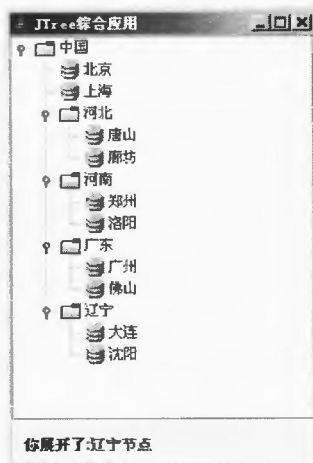


图 11-16 更改后的叶节点

11.8.6 项目总结

在本项目中，综合使用了 JTree、DefaultTreeModel、DefaultMutableTreeNode、TreePath 和 DefaultTreeCellRenderer 对与树有关的事件进行了处理。可以看到，在 NetBeans 中开发具有树状结构的应用程序并不是很复杂的工作。

通过开发该项目，相信读者对与树状结构有关的内容有了更加深入的了解。在以后开发项目的过程中可以在适当的场合选择使用树状结构。

11.9 小结

本章通过具体的实例讲述了如何使用 Swing 提供的各种树状结构控件，包括 JTree、树模型、树路径、树节点等内容。可以看到，在 NetBeans 中开发使用树状结构的程序非常方便，只需要通过鼠标的拖曳即可完成大部分界面的开发，而不需要手工编写太多的代码。

第 12 章

表格的高级应用

表格是一种非常重要的数据组织形式，在用户界面中是非常常见的。表格也是一种最普遍的资料查看形式，非常便于数据记录的分类和挑选。Swing 提供了 `JTable` 控件，使用其可以方便地创建具有表格结构的程序界面。

在本章中将对与表格有关的内容展开讨论，主要包括如下内容：

- 一个使用表格的简单程序；
- `JTable` 控件介绍；
- 表格模型；
- 表格列；
- 表格列模型；
- 与表格有关的事件；
- 表格绘制器；
- 在 NetBeans 中开发使用表格的程序。

12.1 一个使用表格的简单程序

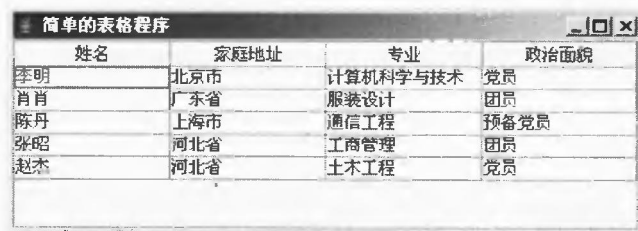
可以把表格想象成为一种二维数据。事实上，`JTable` 的一个构造器可以以 `Object[][]` 为参数，将该二维数组的内容以表格的方式分行、分列显示。

本节中将给出一个使用表格的简单程序，代码如下（`SimpleTable.java`）。此程序的代码非常简单。它以 `String[][]` 类型参数作为表格数据，以 `String[]` 类型参数作为表格的标题栏。需要注意的是：表格（`JTable`）一定要被添加到滚动窗格（`JScrollPane`）中才能正常显示，如果直接使用则可能显示不正常。

```
package org.netbeans.swing.jtable;
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
```

```
public class SimpleTable extends JFrame {
    public SimpleTable() {
        JTable myJTable=new JTable(new String[][]{{"李明","北京市","计算机科学与技术","党员"},
                                                    {"肖肖","广东省","服装设计","团员"},
                                                    {"陈丹","上海市","通信工程","预备党员"},
                                                    {"张昭","河北省","工商管理","团员"},
                                                    {"赵杰","河北省","土木工程","党员"}},
                                     new String[]{"姓名","家庭地址","专业","政治面貌"});
        JScrollPane jsp=new JScrollPane(myJTable);
        this.add(jsp,BorderLayout.CENTER);
        this.setTitle("简单的表格程序");
        this.setBounds(100,100,450,160);
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[]){
        new SimpleTable();
    }
}
```

编译并运行程序，结果如图 12-1 所示。



姓名	家庭地址	专业	政治面貌
李明	北京市	计算机科学与技术	党员
肖肖	广东省	服装设计	团员
陈丹	上海市	通信工程	预备党员
张昭	河北省	工商管理	团员
赵杰	河北省	土木工程	党员

图 12-1 程序运行结果

在上面的程序中，只是向构造器传递了表格数据（String[][]对象）和列标题（String[]对象）作为表格的数据模型，其余工作皆由 JTable 完成。在默认情况下，该程序可以选中多行，编辑单元格并且监听各类事件。对于表格的数据模型以及 JTable 类的详细介绍，会在后面的章节中进行。

12.2 JTable 控件介绍

前一节给出了一个使用表格的简单程序，本节将对如何创建使用 JTable 创建表格展开讨论，主要包括如下内容：

- JTable 类简介；
- NetBeans 中 JTable 的使用。

12.2.1 JTable 类简介

javax.swing.JTable 扩展了 javax.swing.JComponent，是一种标准的 Swing 控件。与其他大

多数 Swing 控件一样, JTable 采用了 Swing MVC 模式。使用 JTable 类创建的仅是数据的视图而没有包含真正的数据, 其数据由其模型管理。

JTable 控件用于显示一个二维对象表格。与其他的 Swing 控件相比, JTable 具有其固有的复杂性, 但是它也是设计比较成功的控件。JTable 控件将表格复杂性隐藏起来, 通过编写很少的几行代码就可以建立功能完善、运行特性丰富的表格。

下面代码创建了一个具有指定标题栏以及指定内容的 JTable 对象。

```
String [ ] [ ] content=new String[][]{{"admin","admin"}, {"system",
"manager"}, {"terry", "123456"}};
String [ ] title=new String[]{"用户名","密码"};
JTable myJTable=new JTable(content,title);
```

除了上面用到的构造器之外, JTable 还提供了其他几个构造器, 表 12-1 列出了这些构造器及其说明。

表 12-1 JTable 常用构造器及说明

构造器	说明
JTable()	构造默认的 JTable, 使用默认的数据模型、默认的列模型和默认的选择模型对其进行初始化
JTable(TableModel dm)	构造 JTable, 使用 dm 数据模型、默认的列模型和默认的选择模型对其进行初始化
JTable(TableModel dm, TableColumnModel cm)	构造 JTable, 使用 dm 数据模型、cm 列模型和默认的选择模型对其进行初始化
JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm)	构造 JTable, 使用 dm 数据模型、cm 列模型和 sm 选择模型对其进行初始化
JTable(int numRows, int numColumns)	使用 DefaultTableModel 构造具有空单元格的 numRows 行和 numColumns 列的 JTable。列名称采用"A"、"B"、"C"的形式
JTable(Vector rowData, Vector columnNames)	构造 JTable, 用来显示 Vectors 的 Vector (rowData)中的值, 其列名称为 columnNames。rowData 中包含的 Vectors 应该包含该行的值

JTable 控件本身不具有滚动窗格的功能, 如果要想实现该功能可以将该 JTable 添加到 JScrollPane 中。出现滚动条后可以滚动表格, 但列标题不会滚动到视区的外面。另外, 还可以单击一个列标题并将其拖移到当前列的左边或右边, 这样可以对各个列的位置进行重新安排。

JTable 类提供了大量的方法供开发人员使用, 表 12-2~表 12-5 列出了其中常用的方法及说明。

表 12-2 JTable 类的添加与删除方法

方法	说明
createDefaultColumnsFromModel()	使用 TableModel 接口中定义的 getColumnCount 方法根据数据模型创建默认的表列。在根据模型的信息创建新的列之前, 清除任何现有的列
clearSelection()	取消选中所有已选定的行和列
addRowSelectionInterval(int index0, int index1)	将从 index0 到 index1 (包含) 之间的行添加到当前选择中

续表

方 法	说 明
<code>addColumnSelectionInterval(int index0, int index1)</code>	将从 <code>index0</code> 到 <code>index1</code> (包含) 之间的列添加到当前选择中
<code>removeRowSelectionInterval(int index0, int index1)</code>	取消选中从 <code>index0</code> 到 <code>index1</code> (包括) 的行
<code>removeColumnSelectionInterval(int index0, int index1)</code>	取消选中从 <code>index0</code> 到 <code>index1</code> (包括) 的列
<code>convertColumnIndexToModel(int viewColumnIndex)</code>	将视图中位于 <code>viewColumnIndex</code> 的列索引映射到表模型中的列索引。返回模型中的对应列索引。如果 <code>viewColumnIndex</code> 小于 0, 则返回 <code>viewColumnIndex</code>
<code>convertColumnIndexToView(int modelColumnIndex)</code>	将表模型中位于 <code>modelColumnIndex</code> 的列索引映射到视图中的列索引
<code>addColumn(TableColumn aColumn)</code>	将 <code>aColumn</code> 追加到此 <code>JTable</code> 的列模型所保持的列数组的结尾。如果 <code>aColumn</code> 的列名称为 <code>null</code> , 则将 <code>aColumn</code> 的列名称设置为 <code>getModel().getColumnName()</code> 所返回的名称
<code>removeColumn(TableColumn aColumn)</code>	从此 <code>JTable</code> 的列数组中移除 <code>aColumn</code> 。注: 此方法不从模型中移除数据列; 它只移除负责显示它的 <code>TableColumn</code>
<code>moveColumn(int column, int targetColumn)</code>	将视图中的 <code>column</code> 列移动到当前被 <code>targetColumn</code> 列所占用的位置。 <code>targetColumn</code> 位置上的原有列向左或右移动以留出空间
<code>columnAtPoint(Point point)</code>	返回 <code>point</code> 位置的列索引, 如果结果不在 <code>[0, getColumnCount()-1]</code> 范围内, 则返回 -1
<code>columnAdded(TableColumnModelEvent e)</code>	将列添加到表的列模型时调用
<code>columnRemoved(TableColumnModelEvent e)</code>	从表的列模型中移除列时调用
<code>columnMoved(TableColumnModelEvent e)</code>	重新定位列时调用。如果正在编辑某个单元格则停止编辑并重新绘制该单元格
<code>columnMarginChanged(ChangeEvent e)</code>	当列由于间距的改变而被移动时调用。如果正在编辑某个单元格则停止编辑并重新绘制该单元格
<code>columnSelectionChanged(ListSelectionEvent e)</code>	<code>TableColumnModel</code> 的选择模型更改时调用
<code>valueChanged(ListSelectionEvent e)</code>	行选择更改时调用, 重新绘制来显示新的选择
<code>editingCanceled(ChangeEvent e)</code>	编辑取消时调用。丢弃编辑器对象并再次呈现单元格
<code>editingStopped(ChangeEvent e)</code>	编辑结束时调用。保存更改并丢弃编辑器

表 12-3

JTable 类的获取与设置方法

方 法	说 明
<code>setTableHeader(JTableHeader tableHeader)</code>	将此 <code>JTable</code> 所使用的 <code>tableHeader</code> 设置为 <code>newHeader</code> 。 <code>tableHeader</code> 为 <code>null</code> 是合法的
<code>getTableHeader()</code>	返回此 <code>JTable</code> 所使用的 <code>tableHeader</code>
<code>setRowHeight(int rowHeight)</code>	将所有单元格的高度设置为 <code>rowHeight</code> (以像素为单位), 重新验证并重新绘制它。单元格的高度等于行高减去行间距
<code>getRowHeight()</code>	返回表的行高, 以像素为单位。默认的行高为 16

续表

方 法	说 明
setRowHeight(int row,int row Height)	将 row 的高度设置为 rowHeight, 重新验证并重新绘制它。此行中单元格的高度等于行高减去行间距
getRowHeight(int row)	返回 row 中单元格的高度 (以像素为单位)
setRowMargin(int rowMargin)	设置相邻行中单元格之间的间距
getRowMargin()	获取单元格之间的间隔 (以像素为单位)
setIntercellSpacing(Dimension intercellSpacing)	将 rowMargin 和 columnMargin (单元格之间间距的高度和宽度) 设置为 intercellSpacing
getIntercellSpacing()	返回单元格之间的水平间距和垂直间距
setGridColor(Color gridColor)	将网格线的颜色设置为 gridColor 并重新显示它。默认的颜色取决于外观
getGridColor()	返回用来绘制网格线的颜色。默认的颜色取决于外观
setShowGrid(boolean showGrid)	设置表是否绘制单元格周围的网格线。如果 showGrid 为 true 则绘制网格线; 如果为 false 则不绘制
setShowHorizontalLines(boolean showHorizontalLines)	设置表是否绘制单元格之间的水平线。如果 showHorizontalLines 为 true 则绘制水平线; 如果为 false 则不绘制
setShowVerticalLines(boolean showVerticalLines)	设置表是否绘制单元格之间的垂直线。如果 showVerticalLines 为 true, 则绘制垂直线; 如果为 false, 则不绘制
getShowHorizontalLines()	如果表绘制单元格之间的水平线, 则返回 true, 否则返回 false。默认值为 true
getShowVerticalLines()	如果表绘制单元格之间的垂直线, 则返回 true, 否则返回 false。默认值为 true
setAutoResizeMode(int mode)	当调整表的大小时, 设置表的自动调整模式
getAutoResizeMode()	返回表的自动调整模式
setModel(TableModel dataModel)	将此表的数据模型设置为 newModel
getModel()	返回提供此 JTable 所显示数据的 TableModel
setColumnModel(TableColumn Model columnModel)	将此表的列模型设置为 newModel, 并通知新列模型的侦听器进行注册
getColumnModel()	返回包含此表所有列信息的 TableColumnModel
setSelectionModel(ListSelection Model newModel)	将此表的行选择模型设置为 newModel, 并通知新选择模型的侦听器进行注册
getSelectionModel()	返回用来维持行选择状态的 ListSelectionModel

表 12-4

JTable 类的获取与设置方法续

方 法	说 明
SetAutoCreateColumnsFromModel (boolean autoCreateColumnsFromModel)	设置此表的 autoCreateColumnsFromModel 标志
getAutoCreateColumnsFromModel()	确定表是否要根据模型创建默认的列, 则 setModel 将清除任何现有的列并根据新模型创建新的列
setDefaultRenderer(Class columnClass, TableCellRenderer renderer)	如果在 TableColumn 中未设置渲染器, 则设置要使用的默认单元格渲染器
getDefaultRenderer(Class columnClass)	在 TableColumn 中尚未设置渲染器时, 返回要使用的单元格渲染器

续表

方 法	说 明
setDefaultEditor(Class columnClass, TableCellEditor editor)	如果尚未在 TableColumn 中设置编辑器, 则设置要使用的默认单元格编辑器
getDefaultEditor(Class columnClass)	在 TableColumn 中尚未设置编辑器时, 返回要使用的编辑器
setDragEnabled(boolean b)	设置 dragEnabled 属性, 该属性必须为 true 才能确保启用对此组件的自动拖动处理
getDragEnabled()	获取 dragEnabled 属性的值
setSelectionMode(int selectionMode)	将表的选择模式设置为只允许单个选择、单个连续单元格选择或多个连续选择
setRowSelectionAllowed(boolean rowSelectionAllowed)	设置是否可以选择此模型中的行
getRowSelectionAllowed()	如果可以选择行, 则返回 true
setColumnSelectionAllowed(boolean columnSelectionAllowed)	设置是否可以选择此模型中的列
getColumnSelectionAllowed()	如果可以选择列, 则返回 true
setCellSelectionEnabled(boolean cellSelectionEnabled)	设置此表是否允许同时存在行选择和列选择
getCellSelectionEnabled()	如果同时启用了行选择模型和列选择模型, 则返回 true
getSelectedRow()	返回首个选定行的索引, 如果没有选定的行, 则返回-1
getSelectedColumn()	返回首个选定列的索引, 如果没有选定的列, 则返回-1
getSelectedRows()	返回所有选定行的索引
getSelectedColumns()	返回所有选定列的索引
getSelectedRowCount()	返回选定的行数
getSelectedColumnCount()	返回选定的列数
getCellEditor()	返回单元格编辑器
setCellEditor(TableCellEditor anEditor)	设置 cellEditor 变量
getCellRenderer(int row,int column)	返回适于由行和列所指定单元格的渲染器
setEditingRow(int aRow)	设置 editingRow 变量

表 12-5

JTable 类的其他常用方法

方 法	说 明
selectAll()	选择表中的所有行、列和单元格
editCellAt(int row,int column)	如果 row 和 column 位置的索引在有效范围内并且这些索引处的单元格是可编辑的, 则以编程方式启动该位置单元格的编辑
isEditing()	如果正在编辑单元格, 则返回 true
isRowSelected(int row)	如果指定的索引位于行的有效范围内并且在该索引位置的行被选定, 则返回 true
isColumnSelected(int column)	如果指定的索引位于列的有效范围内并且在该索引位置的列被选定, 则返回 true
isCellSelected(int row,int column)	如果指定的索引位于行和列的有效范围内并且在该指定位置的单元格被选定, 则返回 true

另外, JTable 还可以使用从其父类中继承的各种方法进行相应的操作, 需要的读者可以查看相关的 API, 这里不再详细讲述。

12.2.2 NetBeans 中 JTable 的使用实例

前面几节对与表格相关的内容进行了详细的介绍。本节将通过一个例子, 介绍如何在 Netbeans 中使用 JTable 控件。启动 NetBeans, 按照如下步骤完成该例。

(1) 新建一个名称为 table 的项目, 将其主类名称设置为 org.netbeans.swing.Demo。

(2) 向 table 中添加一个通过 JFrame Form 模版创建类, 名称为 DemoJTable。


(3) 从 “Palette” 窗口向 DemoJTable 中添加一个 JTable 并将其名称修改为 jTableFactory。此时界面如图 12-2 所示。

Title 1	Title 2	Title 3	Title 4

图 12-2 GUI 设计器中内容

提示

JTable 控件本身没有自动滚动的功能, 需要将其添加到 JScrollPane 中才能实现该功能, 而且没有添加到 JScrollPane 中的表格还可能显示不正确。在 NetBeans 中使用 JTable 时, NetBeans 会自动将 JTable 添加到 JScrollPane 中而不需要开发人员编写代码进行添加。

(4) 在对象观察器窗口中选中 “jTableFactory” 节点, 在属性窗口中修改其属性。单击 model 属性右侧的  按钮, 界面如图 12-3 所示。

(5) Column 列中内容为列的索引号, 其内容不可编辑。Title 列中值为表格标题栏的内容。Editable 定义表格列的可编辑性。Type 列中值为表格相应列的类型, 选中 Type 列中的相应内容后, 出现图 12-4 所示内容。

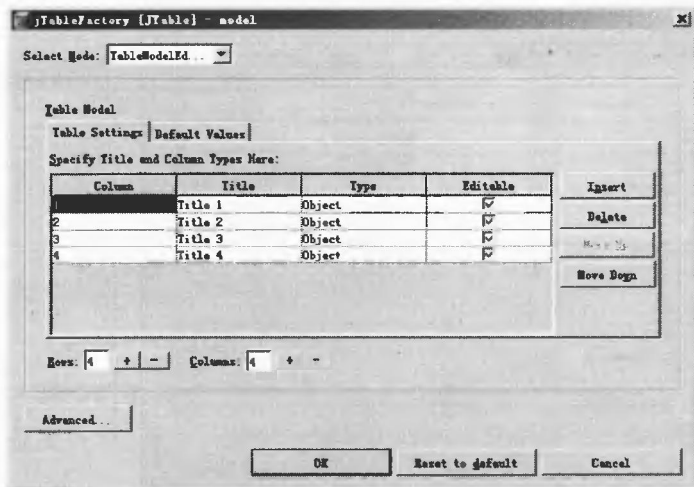


图 12-3 model 属性设置界面



图 12-4 设置 Type 类型

选择了不同的列类型，在程序运行时表格列所对应的编辑器也会有所不同，比如选择 **提示** 了 String 类型，则此列可以输入任何字符串，而如果选择了 Integer 类型，则此列只能输入整数，否则报错。这对于开发来者说是非常方便的。

(6) 可以通过在图 12-4 所示窗口中选中不同的值来修改列的数据类型。修改 Rows 文本框与 Column 文本框的内容，可以设置表格的行数或列数。另外，还可以通过单击窗口右侧的“Insert”与“Delete”按钮添加或删除列。单击“Move Up”与“Move Down”按钮可以调整列的位置。修改如图 12-4 所示窗口中的内容，修改后的窗口如图 12-5 所示。

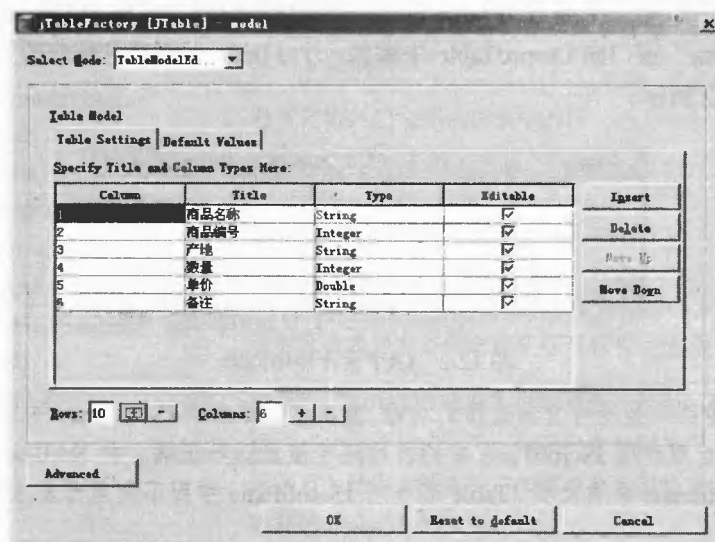


图 12-5 修改内容后的窗口

(7) 在图 12-5 所示窗口中单击“Default Values”标签项。此时界面如图 12-6 所示。

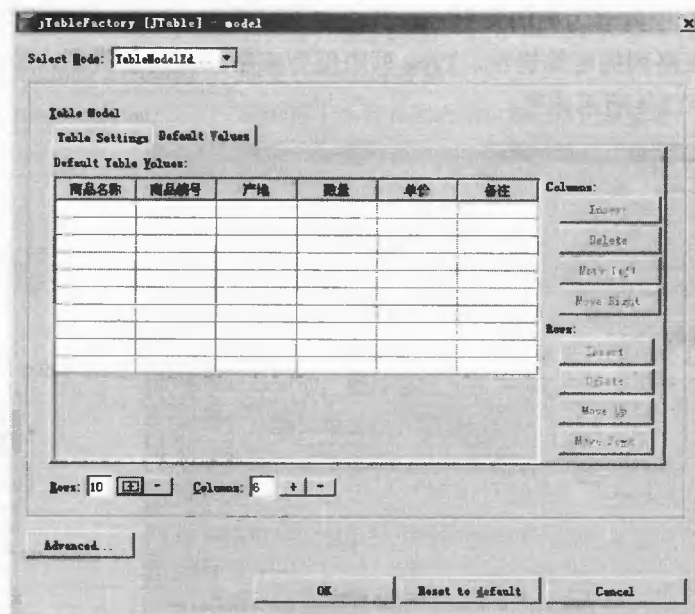


图 12-6 设置表格内容窗口

(8) 可以在图 12-6 所示窗口编辑单元格内容, 该内容为程序运行时表格的初始内容。按照表 12-6 编辑单元格的内容。

表 12-6

表格的初始内容

商品名称	商品编号	产地	数量	单价	备注
电脑	123456	北京	20	4999.0	不可优惠
汽车	321546	上海	15	632565.0	可优惠
手机	985632	韩国	68	1853.0	不可优惠
MP3	897452	杭州	120	368.0	货源充足
主板	897546	台湾	88	620.0	货源充足
显卡	895210	深圳	59	430.0	缺货
CPU	895632	美国	650	1230.0	不可优惠
手表	563241	瑞士	650	95800.0	需要预定
图书	432568	北京	6580	58.0	货源充足
香烟	589623	云南	658	320.0	货源充足

说明

通过单击图 12-6 所示窗口右端的按钮, 还可以进行添加、删除行或列以及移动行或列的位置等。

(9) 编辑单元格内容以后, 单击“OK”按钮完成。此时 GUI 设计器中内容如图 12-7 所示。

(10) 将下列代码添加到 Demo 类的 public static void main(String args[])方法的方法体内。

```
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoJTable().setVisible(true);
    }
});
```

(11) 编译并运行项目, 结果如图 12-8 所示。

商品名称	商品编号	产地	数量	单价	备注
电脑	123456	北京	20	4,999	不可优惠
汽车	321546	上海	15	632,565	可优惠
手机	985632	韩国	68	1,853	不可优惠
MP3	897452	杭州	120	368	货源充足
主板	897546	台湾	88	620	货源充足
显卡	895210	深圳	59	430	缺货
CPU	895632	美国	650	1,230	不可优惠
手表	563241	瑞士	650	95,800	需要预定
图书	432568	北京	6580	58	货源充足
香烟	589623	云南	658	320	货源充足

图 12-7 GUI 设计器中内容

商品名称	商品编号	产地	数量	单价	备注
电脑	123456	北京	20	4,999	不可优惠
汽车	321546	上海	15	632,565	可优惠
手机	985632	韩国	68	1,853	不可优惠
MP3	897452	杭州	120	368	货源充足
主板	897546	台湾	88	620	货源充足
显卡	895210	深圳	59	430	缺货
CPU	895632	美国	650	1,230	不可优惠
手表	563241	瑞士	650	95,800	需要预定
图书	432568	北京	6580	58	货源充足
香烟	589623	云南	658	320	货源充足

图 12-8 程序运行界面

双击相应的单元格, 则该单元格处于编辑状态。如果在类型为数值型的列的单元格中输入非数值型内容, 则单元格的边框变为红色并且拒绝焦点移出该单元格, 图 12-9 显示了这种情况。

从图中可以看到, 在单元格中输入了非法的内容, 则该单元格的边框变为红色并且不能

将焦点移出该单元格。

商品名	商品编号	产地	数量	单价	备注
电脑	123456	北京	ssf78379kjhd	4,999	不可优
汽车	321546	上海	15	632,565	可优惠
手机	985632	韩国	66	1,853	不可优
MP3	897452	杭州	120	368	货源充
主板	897546	台湾	88	620	货源充
显卡	897546	深圳	60	1,500	货源充

图 12-9 拒绝焦点移出示意图

12.3 表格模型——TableModel

上一节中已经提到，JTable 类创建的只是表格的视图，其并不包含真正的数据。一般情况下，JTable 使用实现了 javax.swing.table.TableModel 接口的类对象作为其数据模型。本节将对表格模型展开讨论，主要包括如下内容：

- AbstractTableModel 类；
- DefaultTableModel 类；
- 在 NetBeans 中开发使用表格模型的程序。

12.3.1 AbstractTableModel 类

Swing 提供了两个实现了 TableModel 接口的类，即 AbstractTableModel 类和 DefaultTableModel 类。开发人员可以使用这两个类的对象作为其数据模型。本节将详细介绍 AbstractTableModel 类的使用。DefaultTableModel 类的使用则在下一节中介绍。

TableModel 接口中定义了一些可供开发人员实现的方法，表 12-7 给出了其中常用的方法及其说明。

表 12-7 AbstractTableModel 类的方法及说明

方 法	说 明
getRowCount()	返回该模型中的行数。JTable 使用此方法来确定它应该显示多少行
getColumnCount()	返回该模型中的列数
getColumnName(int columnIndex)	返回 columnIndex 位置的列的名称
getColumnClass(int columnIndex)	针对列中所有的单元格值，返回最具体的超类类型
isCellEditable(int rowIndex, int columnIndex)	如果 rowIndex 和 columnIndex 位置的单元格是可编辑的，则返回 true。否则，在该单元格上调用 setValueAt 不会更改该单元格的值
getValueAt(int rowIndex, int columnIndex)	返回 columnIndex 和 rowIndex 位置的单元格值
setValueAt(Object aValue, int rowIndex, int columnIndex)	将 columnIndex 和 rowIndex 位置的单元格中的值设置为 aValue
addTableModelListener(TableModelListener l)	每当数据模型发生更改时就将一个侦听器添加到被通知的列表中
removeTableModelListener(TableModelListener l)	每当数据模型发生更改时就从被通知的列表中移除一个侦听器

续表

方 法	说 明
findColumn(String columnName)	返回具有给定名称的列
fireTableDataChanged()	通知所有侦听器，表的所有行单元格值可能已更改。行数也可能已更改，并且 JTable 应该从头开始重新绘制该表
fireTableStructureChanged()	通知所有侦听器，表的结构已更改。JTable 应该重新绘制该表格
fireTableRowsInserted(int firstRow,int lastRow)	通知所有侦听器，已插入范围在 firstRow~lastRow 的行
fireTableRowsUpdated(int firstRow, int lastRow)	通知所有侦听器，已更新范围在 firstRow~lastRow 的行
fireTableRowsDeleted(int firstRow,int lastRow)	通知所有侦听器，已删除范围在 firstRow~lastRow 的行
fireTableCellUpdated(int row,int column)	通知所有侦听器，已更新[row, column]处的单元格值
fireTableChanged(TableModelEvent e)	将给定的通知事件转发到所有将自身注册为此表模型的侦听器的 TableModelListeners

AbstractTableModel 类是一个抽象类，实现了 TableModel 接口中的大多数方法，但它将最重要的部分留给开发人员完成，是开发人员创建自己表格模型的起点。如果开发人员想要建立自己的表格模型，应该选择扩展 AbstractTableModel 类，而不是 DefaultTableModel 类。AbstractTableModel 类中 3 个未实现的方法分别是：

```
public abstract int getColumnCount();
public abstract int getRowCount();
public abstract Object getValueAt(int row,int col);
```

上述 3 个方法用于创建开发人员自己的表格模型，从而更好地配置需要显示的各种数据类型。下面给出了一个扩展了 AbstractTableModel 类的完整代码（ExtendsAbstractTableModel.java），此程序的功能是列出本程序所在目录所有文件的信息。

```
package org.netbeans.swing.jtable;
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
import java.io.File;
public class ExtendsAbstractTableModel extends AbstractTableModel{
    String []title;
    Object content[][];
    Class columnType[];
    public ExtendsAbstractTableModel () {
        columnType=new Class[]
        {String.class,Number.class,Boolean.class,Boolean.class}; //设置列的类型
        title=new String[]{"文件名","文件大小","可读","可写"};
        this.initialContent(); //调用 initialContent 方法
    }
    //重写 getColumnCount 方法
```

```
public int getColumnCount() {
    return this.title.length;
}
//重写 getRowCount 方法
public int getRowCount() {
    return this.content.length;
}
//重写 getValueAt 方法
public Object getValueAt(int row,int col) {
    return content[row][col];
}
//重写 getColumnName 方法, 如果不重写该方法, 则默认的列名为 A,B,C...
public String getColumnName(int col) {
    return title[col];
}
//重写 getColumnClass 方法
public Class getColumnClass(int col) {
    return columnType[col];
}

//该方法用来初始化表格的内容
public void initialContent(){
    File tempEmptyFile=new File(".");//创建空文件
    String []allFileName=tempEmptyFile.list();//获取文件所在目录的所有文件
    content=new Object[allFileName.length][title.length];
    for(int i=0;i<allFileName.length;i++){
        File tempFile=new File(allFileName[i]);
        content[i][0]=new String(tempFile.getName());
        content[i][1]=new Long(tempFile.length());
        content[i][2]=new Boolean(tempFile.canRead());
        content[i][3]=new Boolean(tempFile.canWrite());
    }
}
//end initialContent
public static void main(String args[]){
    ExtendsAbstractTableModel myModel=new ExtendsAbstractTableModel();
    JTable myJTable=new JTable(myModel);//使用 myModel 创建表格
    JFrame userModel=new JFrame();
    JScrollPane jsp=new JScrollPane(myJTable);
    userModel.add(jsp,BorderLayout.CENTER);
    userModel.setTitle("简单的表格程序");
    userModel.setBounds(100,100,450,160);
    userModel.setVisible(true);
    userModel.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
//end main
}
```

编译并运行程序, 结果如图 12-10 所示。

文件名	文件大小	可读	可写
0082.jpg	25075	✓	✓
0083.jpg	8606	✓	✓
0084.jpg	9185	✓	✓
0086.jpg	3884	✓	✓
0159.jpg	7910	✓	✓
0160.jpg	24850	✓	✓
2.jpg	20635	✓	✓
3.jpg	7521	✓	✓
4.jpg	791	✓	✓
4.png	791	✓	✓
5.jpg	131436	✓	✓
ExtendsAbstractTableModel.java	2074	✓	✓
org	0	✓	✓
SimpleTable.java	1512	✓	✓

图 12-10 程序运行界面

可以不重写 `getColumnName` 与 `getColumnClass` 方法。如果不重写 `getColumnName` 方法, 则表格使用默认的 A、B、C、D... 作为标题。如果不重写 `getColumnClass` 方法, 则可读与可写列中使用字符串 `true` 或 `false` 表示。

12.3.2 DefaultTableModel 类

`DefaultTableModel` 扩展了 `AbstractTableModel`。一般情况下, 其使用对象数组或由多个 `Vector` 对象构成的 `Vector` 来存储单元格的值。默认情况下, `DefaultTableModel` 模型假定每一个单元格都可以编辑。以下代码使用 `Vector` 来创建 `DefaultTableModel` 对象:

```
Vector title=new Vector();//title 用来存放标题内容
title.add("姓名");
title.add("学号");
title.add("年级");
Vector firstLine=new Vector();//firstLine 用来存放第一行的内容
firstLine.add("王烨");
firstLine.add("123456");
firstLine.add("三年级");
Vector secondLine=new Vector();//secondLine 用来存放第二行的内容
secondLine.add("郝静");
secondLine.add("985647");
secondLine.add("三年级");
Vector thirdLine=new Vector();//thirdLine 用来存放第三行的内容
thirdLine.add("王小森");
thirdLine.add("563288");
thirdLine.add("一年级");
//创建表示内容的 Vector 对象并将表示行内容的的 Vector 对象添加到其中
Vector content=new Vector();
content.add(firstLine);
content.add(secondLine);
content.add(thirdLine);
//使用包含内容的 Vector 对象和包含标题的 Vector 对象创建表格模型
DefaultTableModel myDefaultTableModel=new DefaultTableModel(content,title);
```

除了上面使用的构造器，DefaultTableModel 类还提供了其他几个构造器，表 12-8 列出了其他几个构造器及说明。

表 12-8 DefaultTableModel 类的构造器及说明

构造器	说明
DefaultTableModel()	构造默认的 DefaultTableModel，它是一个零列零行的表
DefaultTableModel(int rowCount, int columnCount)	构造具有 rowCount 和 columnCount 个 null 对象值的 DefaultTableModel
DefaultTableModel(Vector columnNames, int rowCount)	构造 DefaultTableModel，它的列数与 columnNames 中元素的列数相同，并具有 rowCount 个 null 对象值。每列的名称都取自 columnNames 向量
DefaultTableModel(Object[] columnNames, int rowCount)	构造 DefaultTableModel，它的列数与 columnNames 中元素的列数相同，并具有 rowCount 个 null 对象值。每列的名称都取自 columnNames 数组
DefaultTableModel(Object[][] data, Object[] columnNames)	构造 DefaultTableModel。Object[][] 数组中的第一个索引是行索引，第二个索引是列索引


DefaultTableModel 提供了一些方法供开发人员使用，表 12-9 给出了其中一些常用的方法及其说明。

表 12-9 DefaultTableModel 的常用方法及说明

方法	说明
getDataVector()	返回由多个 Vector 组成的 Vector，它包含表的数据值
setDataVector(Vector dataVector, Vector columnIdentifiers)	用新的行 Vector (dataVector) 替换当前的 dataVector 实例变量
setDataVector(Object[][] dataVector, Object[] columnIdentifiers)	用数组 dataVector 中的值替换 dataVector 实例变量中的值
newRowsAdded(TableModelEvent e)	确保新行的列数正确。这是通过使用 Vector 中的 setSize 方法来完成，该方法在 Vector 太长时将其截短，在 Vector 太短时追加 null
setRowCount(int rowCount)	设置模型中的行数。如果新的大小大于当前大小，则将新行添加到该模型的结尾，如果新的大小小于当前大小，则丢弃索引 rowCount 及其之后的所有行
addRow(Vector rowData)	添加一行到模型的结尾，内容由 Vector 确定。如果未指定 rowData，则新行将包含 null 值
addRow(Object[] rowData)	添加一行到模型的结尾，内容由 Object 数组确定。如果未指定 rowData，则新行将包含 null 值
insertRow(int row, Vector rowData)	在模型中的 row 位置插入一行。内容由 Vector 确定。如果未指定 rowData，则新行将包含 null 值
insertRow(int row, Object[] rowData)	在模型中的 row 位置插入一行。内容由 Vector 确定。如果未指定 rowData，则新行将包含 null 值
moveRow(int start, int end, int to)	将 start (包含) ~ end (包含) 范围中的一行或多行移到模型中的 to 位置。进行移动操作后，原先索引为 start 的行的索引将变为 to
removeRow(int row)	移除模型中 row 位置的行
setColumnCount(int columnCount)	设置模型中的列数。如果新大小大于当前大小，则将新列添加到模型结尾并使其具有 null 单元格值。如果新大小小于当前大小，则将丢弃索引 columnCount 处及其之后的所有列

续表

方 法	说 明
<code>addColumn(Object columnName)</code>	将一列添加到模型中。新列的标识符将为 <code>columnName</code> , 可能为 <code>null</code>
<code>addColumn(Object columnName, Vector columnData)</code>	将一列添加到模型中
<code>getRowCount()</code>	返回此数据表中的行数
<code>getColumnCount()</code>	返回此数据表中的列数
<code>getColumnName(int column)</code>	返回列名称
<code>isCellEditable(int row,int column)</code>	返回单元格的可编辑性, 无论参数值是多少都返回 <code>true</code>
<code>getValueAt(int row,int column)</code>	返回 <code>row</code> 和 <code>column</code> 处单元格的属性值
<code>setValueAt(Object aValue,int row, int column)</code>	使用 <code>aValue</code> 设置 <code>column</code> 和 <code>row</code> 处单元格的对象值

 **提示** DefaultTableModel 对于开发使用表格的程序是非常重要的, 熟练的掌握这些方法的使用, 对于提高编程效率是很有帮助的。

12.3.3 在 NetBeans 中开发使用表格模型的程序

前面两个小节对表格模型进行了详细的介绍, 本节将通过一个例子, 介绍如何在 NetBeans 中使用表格模型。打开前面创建的项目 `table`, 按如下步骤完成该例。

- (1) 向 `table` 中添加通过 `JFrame Form` 模版创建的名称为 `DemoTableModel` 的类。
- (2) 向 `DemoTableModel` 中添加一个 `JTable`, 将其名称修改为 `jTableUseModel`, 修改其 `model` 属性。在图 12-3 所示的窗口中选中“Select Model”下拉列表框中的“Form Connection”选项, 此时窗口如图 12-11 所示。
- (3) 选中“User Code”单选按钮, 将下列代码添加到其右侧的文本框中:

```
this.initialTableModel();
```

- `initialTableModel` 是 `DemoTableModel` 中一个自定义的方法, 用来创建并初始化 `DefaultTableModel` 对象并返回, 后面将会给出该方法。
 - 在“User Code”中填上“`this.initialTableModel();`”表示此表格使用 `initialTableModel` 方法生成的 `DefaultTableModel` 对象作为其模型。
- (4) 向 `DemoTableModel` 中添加 4 个 `JButton`, 按照如表 12-10 所示的内容修改其属性。

表 12-10 属性与值对照表

控 件 名 称	text 属性值	控 件 名 称	text 属性值
<code>jButtonAddRow</code>	添加行	<code>jButtonDeleteRow</code>	删除行
<code>jButtonAddColumn</code>	添加列	<code>jButtonDeleteColumn</code>	删除列

- (5) 此时 GUI 设计器中的界面如图 12-12 所示。
- (6) 向 `DemoTableModel` 中添加名为“`public DefaultTableModel initialTableModel`”的方法。该方法使用给定的内容初始化表格的模型。将下列代码添加到该方法中:

```
Vector title=new Vector();//title 向量用来存放标题的内容
title.add("第一列");
```



```

title.add("第二列");
title.add("第三列");
Vector firstRow=new Vector();//firstRow 向量用来存放第一行的内容
firstRow.add("11111");
firstRow.add("11111");
firstRow.add("11111");
Vector secondRow=new Vector();//secondRow 向量用来存放第二行的内容
secondRow.add("22222");
secondRow.add("22222");
secondRow.add("22222");
Vector thirdRow=new Vector();//thirdRow 向量用来存放第三行的内容
thirdRow.add("33333");
thirdRow.add("33333");
thirdRow.add("33333");
//表示各行内容的向量添加到 content 向量中, content 向量用来存放表格中的数据
Vector content=new Vector();
content.add(firstRow);
content.add(secondRow);
content.add(thirdRow);
//使用包含内容的向量和包含标题的向量创建表格模型
DefaultTableModel myDefaultTableModel=new
DefaultTableModel(content,title);
return myDefaultTableModel;

```

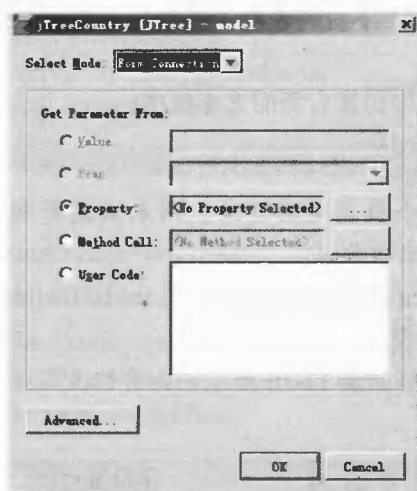


图 12-11 设置 model 窗口属性



图 12-12 GUI 设计器中内容

(7) 分别为 4 个按钮添加 ActionEvent 事件的处理方法, 并按照如下所述向相应方法的方法体内添加代码。

● 向 jButtonAddRowActionPerformed 方法中添加如下代码。

```

//获取表格模型
DefaultTableModel myModel=((DefaultTableModel) jTableUseModel.getModel());
//创建一个用来存储新添加行内容的 Vector

```



```

Vector newRow=new Vector();
int columnNum=myModel.getColumnCount();//获取表格的列数
for(int i=0;i<columnNum;i++){
    newRow.add("");, 将该新行的内容设置为空
}
myModel.getDataVector().add(newRow);
//通知所有的监听器表格模型已经改变, 以更新表格视图
//如果没有此句代码添加了的新行不会出现
((DefaultTableModel) jTableUseModel.getModel()).fireTableStructureChanged();

```

● 向 `jButtonDeleteRowActionPerformed` 方法中添加如下代码:

```

//获取表格的当前模型
DefaultTableModel myModel=(DefaultTableModel) jTableUseModel.getModel();
int rowNum=myModel.getRowCount();//获取表格的行数
if(rowNum>0)
myModel.removeRow(rowNum-1);//删除最后一行

```

● 向 `jButtonAddColumnActionPerformed` 方法中添加如下代码:

```

DefaultTableModel myModel=(DefaultTableModel) jTableUseModel.getModel();
Vector newColumn=new Vector();
int columnCount=myModel.getColumnCount()+1;//获取要插入的列索引
myModel.addColumn("第"+columnCount+"列");//将添加类的标题设置为“第 XXX 列”

```

● 向 `jButtonDeleteColumnActionPerformed` 方法中添加如下代码:

```

int columnCount=myModel.getColumnCount()-1;
if(columnCount>0)
myModel.setColumnCount(columnCount);//设置表格的列数, 最后一列会被清除

```

(8) 将下列代码添加到 `Demo` 类的 `public static void main(String args[])` 方法中:

```

java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoTableModel().setVisible(true);
    }
});

```

(9) 编译并运行项目, 结果如图 12-13 所示。

(10) 单击“添加行”按钮后, 会在表格中添加一空行。单击 4 次“添加行”按钮后的效果如图 12-14 所示。



图 12-13 程序运行界面



图 12-14 添加行

(11) 单击“添加列”按钮后，会在表格中添加一空列，其标题为“第 XXX 列”。单击 3 次“添加列”按钮后的效果如图 12-15 所示。



图 12-15 添加列

另外，单击“删除行”或“删除列”按钮，可以删除表格的最后一行或者最后一列。

12.4 表格列——TableColumn

Swing 表格的基本单位不是单元格而是列。实际应用中，大多数的列表示适用于列中所有记录的某一类信息，列中的所有记录具有一致的数据类型。TableColumn 类是创建列的起点，是访问列中所有内容的入口。

java.swing.table.TableColumn 直接扩展了 java.lang.Object。TableColumn 可以用来表示 JTable 中列的所有属性，如宽度、大小可调整性、最小和最大宽度等。此外 TableColumn 还为显示和编辑此列中值的渲染器和编辑器提供了位置信息。以下代码创建了一个简单的 TableColumn 对象。

```
TableColumn myTableColumn=new TableColumn();
```

创建 TableColumn 对象以后，可以使用 JTable 类的 addColumn 方法向表格中添加列。

```
myJTable. addColumn (myTableColumn); // myJTable 为已有的 JTable 对象
```

除了使用上面给出的由空构造器创建的 TableColumn 对象，还可以使用 TableColumn 类提供的其他几个构造器创建 TableColumn 对象，表 12-11 列出了这些构造器及说明。

表 12-11 TableColumn 构造器及说明

构造器	说明
TableColumn(int modelIndex)	用指定的 modelIndex 创建一个空列，其他属性均为默认值
TableColumn(int modelIndex,int width)	用指定的 modelIndex 以及宽度（单位像素）创建一个空列，其最小宽度和最大宽度仍取默认值
TableColumn(int modelIndex,int width, TableColumnRenderer cellRenderer, TableColumnEditor cellEditor)	创建并初始化具有 modelIndex 的 TableColumn 实例。所有 TableColumn 构造器最终都调用该构造器创建 TableColumn 对象

创建 TableColumn 对象以后，可以使用 TableColumn 类提供的方法对其相应的属性进行

操作，表 12-12 给出了其中常用的一些方法及其说明。

表 12-12 `TableColumn` 的常用方法及说明

方 法	说 明
<code>setModelIndex(int modelIndex)</code>	设置此列的模型索引。模型索引是模型中此 <code>TableColumn</code> 将要显示的列的索引。 <code>TableColumn</code> 在视图中移动时，该模型索引保持不变
<code>getModelIndex()</code>	返回此列的模型索引
<code>setHeaderValue(Object headerValue)</code>	设置 <code>Object</code> ，将使用其字符串表示形式作为 <code>headerRenderer</code> 的值
<code>getHeaderValue()</code>	返回用作标题渲染器值的 <code>Object</code>
<code>setHeaderRenderer(TableCellRenderer headerRenderer)</code>	将 <code>TableColumn</code> 头的 <code>TableCellRenderer</code> 设置为 <code>headerRenderer</code>
<code>getHeaderRenderer()</code>	返回用于绘制 <code>TableColumn</code> 头的 <code>TableCellRenderer</code>
<code>setCellRenderer(TableCellRenderer cellRenderer)</code>	设置 <code>JTable</code> 用于绘制此列各值的 <code>TableCellRenderer</code>
<code>getCellRenderer()</code>	返回 <code>JTable</code> 用于绘制此列各值的 <code>TableCellRenderer</code>
<code>setCellEditor(TableCellEditor cellEditor)</code>	设置编辑此列中单元格时所用的编辑器
<code>getCellEditor()</code>	返回 <code>JTable</code> 用于编辑此列各值的 <code>TableCellEditor</code>
<code>getWidth()</code>	返回该 <code>TableColumn</code> 的宽度。默认宽度为 75
<code>setPreferredWidth(int preferredWidth)</code>	将此列的首选宽度设置为 <code>preferredWidth</code> 。如果 <code>preferredWidth</code> 超出最小或最大宽度，则将其调整为合适的界限值
<code>getPreferredWidth()</code>	返回 <code>TableColumn</code> 的首选宽度。默认首选宽度为 75
<code>setMinWidth(int minWidth)</code>	将 <code>TableColumn</code> 的最小宽度设置为 <code>minWidth</code> ；如果当前宽度和首选宽度小于此值，也对其进行调整
<code>getMinWidth()</code>	返回 <code>TableColumn</code> 的最小宽度。不能通过用户或编程的方式将 <code>TableColumn</code> 的宽度设置为小于此值。默认的 <code>minWidth</code> 为 15
<code>setMaxWidth(int maxWidth)</code>	将 <code>TableColumn</code> 的最大宽度设置为 <code>maxWidth</code> ；如果宽度和首选宽度大于此值，也对其进行调整
<code>getMaxWidth()</code>	返回 <code>TableColumn</code> 的最大宽度。不能通过用户或编程的方式将 <code>TableColumn</code> 的宽度设置为大于此值
<code>setResizable(boolean isResizable)</code>	设置是否可以调整此列的大小
<code>getResizable()</code>	如果允许用户调整 <code>TableColumn</code> 的宽度，则返回 <code>true</code> ；否则返回 <code>false</code>
<code>setWidthToFit()</code>	调整 <code>TableColumn</code> 的大小，以适合其头单元格的宽度。如果头渲染器为 <code>null</code> （默认情况如此），则此方法不执行任何操作
<code>addPropertyChangeListener(PropertyChangeListener listener)</code>	向侦听器列表中添加一个 <code>PropertyChangeListener</code> 。该侦听器是为所有属性注册的
<code>removePropertyChangeListener(PropertyChangeListener listener)</code>	从侦听器列表中移除一个 <code>PropertyChangeListener</code> 。所移除的 <code>PropertyChangeListener</code> 是为所有属性注册的

12.5 表格列模型

表格列模型是开发使用表格的程序时经常用到的一种控件，本节将会对表格列模型进行介绍，主要包括如下内容：

- 默认表格列模型介绍;
- 默认表格列模型的常用方法说明;
- 一个使用表格列模型的简单程序。

12.5.1 默认表格列模型

Swing 程序包中的 `TableColumnModel` 接口封装了与列模型有关的信息, 如列间距、列是否可选等信息, 并且可以对表格的列模型进行管理。

一般情况下, 都会使用实现了 `TableColumnModel` 接口的类 `DefaultTableColumnModel` 来对列模型进行管理, 并且如果没有为表格指定列模型, 则由该类的对象充当默认的列模型。

说明 `DefaultTableColumnModel` 是创建列模型的良好起点, 开发人员可以通过覆盖其中的方法来实现自己的列模型。

`DefaultTableColumnModel` 只提供了一个空构造器, 下列代码使用该构造器创建了一个 `DefaultTableColumnModel` 对象。

```
DefaultTableColumnModel myColumnModel=new DefaultTableColumnModel();
```

12.5.2 默认表格列模型的常用方法说明


创建 `DefaultTableColumnModel` 对象以后, 可以使用 `DefaultTableColumnModel` 类提供的方法进行相应的操作, 表 12-13 列出了其中一些常用的方法及说明。

表 12-13 `DefaultTableColumnModel` 的常用方法及说明

方 法	说 明
<code>addColumn(TableColumn aColumn)</code>	将 <code>aColumn</code> 追加到 <code>tableColumns</code> 数组的结尾
<code>removeColumn(TableColumn column)</code>	从 <code>tableColumn</code> 数组删除 <code>column</code> 。如果 <code>column</code> 不在该表各列的列表中, 则此方法不执行任何操作
<code>moveColumn(int columnIndex, int newIndex)</code>	将 <code>columnIndex</code> 位置的列和标题移到 <code>newIndex</code> 。原来在 <code>columnIndex</code> 位置的列现在可以在 <code>newIndex</code> 处找到
<code>setColumnMargin(int newMargin)</code>	将列空白设置为 <code>newMargin</code>
<code>getColumnCount()</code>	返回 <code>tableColumns</code> 数组中的列数
<code>getColumns()</code>	返回该模型中所有列的一个 <code>Enumeration</code>
<code>getColumnIndex(Object identifier)</code>	返回 <code>tableColumns</code> 数组中第一列的索引
<code>getColumn(int columnIndex)</code>	返回 <code>columnIndex</code> 处列的 <code>TableColumn</code> 对象
<code>getColumnMargin()</code>	返回 <code>TableColumn</code> 的空白宽度。默认的 <code>columnMargin</code> 为 1
<code>getTotalColumnWidth()</code>	返回所有列的总宽度
<code>setSelectionModel(ListSelectionModel newModel)</code>	将此 <code>TableColumnModel</code> 的选择模型设置为 <code>newModel</code>
<code>getSelectionModel()</code>	返回用来维护列选择状态的 <code>ListSelectionModel</code>
<code>setColumnSelectionAllowed(boolean flag)</code>	设置是否允许列选择, 默认值为 <code>false</code>
<code>getColumnSelectionAllowed()</code>	如果允许列选择, 则返回 <code>true</code> , 否则返回 <code>false</code> , 默认值为 <code>false</code>
<code>getSelectedColumns()</code>	返回包含选定列的数组

续表

方 法	说 明
getSelectedColumnCount()	返回选定的列数
addColumnModelListener (TableColumnModelListener x)	添加表列模型事件的侦听器
removeColumnModelListener (TableColumnModelListener x)	移除表列模型事件的侦听器
getColumnModelListeners()	获取在此列模型上注册的所有列模型侦听器列表

 **说明** 与表格模型一样，列模型对于开发使用表格的应用程序也是非常重要的。熟练的使用上述方法，会大大的提高开发速度。

下面给出了使用 DefaultTableColumnModel 类的方法进行操作的示例：

- 使用 getColumn(int columnIndex)方法获取 columnIndex 处列的 TableColumn 对象，代码如下：

```
myTableModel.getColumnCount();
```

- 使用 getTotalColumnWidth()方法获取所有列的总宽度，代码如下：

```
myTableModel.getTotalColumnWidth();
```

 **说明** myTableModel 为 DefaultTableColumnModel 类的对象句柄。

12.5.3 一个使用表格列模型的简单程序

前面两个小节对默认表格列模型进行了简单介绍，本节开发了一个使用默认表格列模型的简单程序，代码如下 (UserTableColumnModel.java)：

```
package org.netbeans.swing.jtable;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
public class UserTableColumnModel extends JFrame{
    DefaultTableModel myDTM;
    SortColumnModel mySCM;
    JTable myJTable;
    JScrollPane myJSP;
    public UserTableColumnModel() {
        //使用指定的内容创建表格模型
        myDTM=new DefaultTableModel(new String [][]{
            {"first","second","third"},
            {"fourth","five","six"}},
            new String[]{"C","B","A"});
        //创建自定义的列模型对象
        mySCM=new SortColumnModel();
        //使用表格模型与列模型创建 JTable
```

```

myJTable=new JTable(myDTM,mySCM);
//使用 TableModel 接口中定义的 getColumnCount 方法根据列模型创建表列
myJTable.createDefaultColumnsFromModel();
myJSP=new JScrollPane(myJTable);
this.add(myJSP,BorderLayout.CENTER);
this.setTitle("使用列模型");
this.setBounds(100,100,450,160);
this.setVisible(true);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
class SortColumnModel extends DefaultTableColumnModel{
    //重写 addColumn 方法
    public void addColumn(TableColumn tc) {
        super.addColumn(tc); //调用父类的 addColumn 方法
        int newIndex=this.sortedIndexOf(tc);
        if(newIndex!=tc.getModelIndex()){//将列移动到 newIndex 指定的位置
            moveColumn(tc.getModelIndex(),newIndex);
        }
    }
    //自定义方法
    public int sortedIndexOf(TableColumn tc){
        //该方法按照列标题的字母进行排序,并且返回其索引号
        int stop=this.getColumnCount();
        String name=tc.getHeaderValue().toString();
        for(int i=0;i<stop;i++){//返回一个列的索引号码
            if(name.compareTo(getColumn(i).getHeaderValue().toString())<=0){
                return i;
            }
        } //end if
        return stop;
    } // end sortedIndexOf
} //end SortColumnModel
public static void main(String args[]){
    new UserTableColumnModel();
}
}

```

编译并运行程序,结果如图 12-16 所示。

A	B	C
third	second	first
six	five	fourth

图 12-16 程序运行结果

从图中可以看到,虽然在表格模型中将表格的标题依次设置为“C”、“B”、“A”,但使

用列模型以后，列仍按照标题的字母顺序进行排列。

12.6 与表格有关的事件

前面几节讨论了与表格和表格模型有关的内容，本节将对与表格有关的事件展开讨论，主要包括如下内容：

- 表格常用事件及处理方法；
- 在 NetBeans 中开发处理表格事件的程序实例。

12.6.1 表格常用事件及处理方法

表格主要触发 3 类事件：列模型事件 `TableColumnModelEvent`、表格模型事件 `TableModelEvent` 和选择事件 `ListSelectionEvent`。

当用户（或程序）更改列模型时，比如移动、添加或删除列，则会产生 `TableColumnModelEvent` 事件。`TableColumnModelEvent` 对象封装了一些与列有关的信息，表 12-14 列出了获取这些信息的方法及其说明。

表 12-14 `TableColumnModelEvent` 的方法及说明

方 法	说 明
<code>getFromIndex()</code>	返回更改模型以前该列的索引
<code>getToIndex()</code>	返回更改模型以后该列的索引

`TableColumnModelEvent` 事件使用实现了 `TableColumnModelListener` 接口的类对象作为监听器，该接口中定义了一些方法来处理相应的动作，表 12-15 列出了这些方法及其说明。

表 12-15 `TableColumnModelListener` 接口的方法及说明

方 法	说 明
<code>columnAdded(TableColumnModelEvent e)</code>	告知侦听器有一列被添加到模型中
<code>columnRemoved(TableColumnModelEvent e)</code>	告知侦听器有一列从模型中移除
<code>columnMoved(TableColumnModelEvent e)</code>	告知侦听器有一列被重新调整位置
<code>columnMarginChanged(ChangeEvent e)</code>	告知侦听器因为页边空白更改，有一列被移除
<code>columnSelectionChanged(ListSelectionEvent e)</code>	告知侦听器 <code>TableColumnModel</code> 的选择模型发生变更

提示 `TableColumnModelListener` 不仅可以处理 `TableColumnModelEvent` 事件，还可以处理与表格列模型有关的 `ListSelectionEvent` 和 `ChangeEvent` 事件。

当用户更改表格模型时，即触发 `TableModelEvent` 事件。`TableModelEvent` 用来通知监听器表格模型已经更改，使用实现了 `TableModelListener` 接口的类对象来作为监听器。

`TableModelEvent` 提供了 5 个构造器，表 12-16 给出了这 5 个构造器及其说明。

表 12-16 `TableModelEvent` 的构造器及说明

构 造 器	说 明
<code>TableModelEvent(TableModel source)</code>	如果表中的所有行数据都发生更改，那么侦听器应该丢弃基于这些行的所有状态，并重新查询 <code>TableModel</code> 来获得新的行数和所有恰当值

续表

构造器	说 明
<code>TableModelEvent(TableModel source, int row)</code>	此数据行已被更新。为了表示具有不同结构的整个新表都已到达, 可使用 <code>HEADER_ROW</code> 作为 <code>row</code> 的值
<code>TableModelEvent(TableModel source, int firstRow, int lastRow)</code>	<code>firstRow</code> 与 <code>lastRow</code> 之间的行被更新
<code>TableModelEvent(TableModel source, int firstRow, int lastRow, int column)</code>	列 <code>column</code> 的 <code>firstRow</code> ~ <code>lastRow</code> 范围内的单元格已被更新
<code>TableModelEvent(TableModel source, int firstRow, int lastRow, int column, int type)</code>	从 (<code>firstRow</code> , <code>column</code>) 到 (<code>lastRow</code> , <code>column</code>) 的单元格已被更改。 <code>type</code> 应该是 <code>INSERT</code> 、 <code>UPDATE</code> 或 <code>DELETE</code> 类型之一

`TableModelEvent` 类提供了一些方法进行相应的操作, 表 12-17 给出了这些方法及其说明。

表 12-17 `TableModelEvent` 常用方法及说明

方 法	说 明
<code>getFirstRow()</code>	返回第一个被更改的行
<code>getLastRow()</code>	返回最后一个被更改的行
<code>getColumn()</code>	返回事件的列
<code>getType()</code>	返回事件类型, 为 <code>INSERT</code> 、 <code>UPDATE</code> 或 <code>DELETE</code> 类型之一

`TableModelListener` 接口中只定义了一个方法: `tableChanged(TableModelEvent e)`。对 `TableModelEvent` 事件的处理代码需要写在该方法的方法体内。

当用户 (或程序) 选取表格中的行、列或者单元格时, 会触发 `ListSelectionEvent` 事件。`ListSelectionEvent` 类封装了与选择事件有关信息, 并提供了一些用于获取这些信息的方法。表 12-18 列出了这些方法及其说明。

表 12-18 `ListSelectionEvent` 的方法及其说明

方 法	说 明
<code>getFirstIndex()</code>	返回第一个选择状态可能发生更改的行的索引
<code>getLastIndex()</code>	返回最后一个选择状态可能已发生更改的行的索引
<code>getValueIsAdjusting()</code>	如果此事件是多个更改事件之一则返回 <code>true</code>
<code>toString()</code>	返回显示并标识此对象的属性的字符串

`ListSelectionEvent` 使用实现了 `ListSelectionListener` 接口的类对象来作为监听器, 其中只定义了一个方法 `valueChanged(ListSelectionEvent e)`, 当选择值发生改变时即调用该方法。

12.6.2 在 NetBeans 中开发处理表格事件的程序实例

前一节讨论了与表格相关的事件, 本节将通过一个例子, 介绍如何在 NetBeans 中开发处理表格事件的程序。

打开前面创建的项目 `table`, 按如下步骤完成该例。

(1) 向 `table` 中添加通过 `JFrame Form` 模版创建名称为 `DemoActionEvent` 的类。

(2) 向 `DemoActionEvent` 中添加一个 `JTable`, 将其名称修改为 `jTableMyTable`。修改其 `model` 属性, 在设置 `model` 属性对话框中将 “User Code” 文本框的值修改为 “`initialTableModel()`”。

(3) 向 `DemoActionEvent` 中添加一个 `JLabel`, 将其名称修改为 `jLabelMessage`, 并将其 `text` 属性的值修改为 “这里将显示动作信息”。

(4) 向 `DemoActionEvent` 中添加一个签名为 `private DefaultTableModel initialTableModel` 的方法, 并将下列代码添加到该方法中:

```
DefaultTableModel myDefaultTableModel=new DefaultTableModel(  
    new String[][]{  
        {"liuyuehui","318320"},  
        {"niyuding","830414"},  
        {"boyingking","123456"}},  
    new String[]{"用户名","密码"});  
  
return myDefaultTableModel;
```

 说明 该方法用来初始化表格的模型。

(5) 向 `DemoActionEvent` 中添加一个名称为 `DealAction` 且实现了 `ListSelectionListener` 与 `TableColumnModelListener` 接口的内部类, 并将下列代码添加到该类的类中:

```
int nowColNum=0;//用来记录当前选中列的索引  
int nowRowNum=0;//用来记录当前选中行的索引  
public void valueChanged(ListSelectionEvent e) {  
    nowRowNum=jTableMyTable.getSelectedRow();  
    this.createMessage();  
}  
public void columnSelectionChanged(ListSelectionEvent e) {  
    nowColNum=jTableMyTable.getSelectedColumn();  
    this.createMessage();  
}  
  
//该方法用来获取并在 jTableMyTable 上显示当前选中的单元格的值,  
public void createMessage() {  
    String ss=(String)jTableMyTable.getValueAt(nowRowNum,nowColNum);  
    DemoActionEvent.this.jLabelMessage.setText("当前选择为第"+(nowRowNum+1)  
        +"行第"+(nowColNum+1)+"列, 值为"+ss);  
}  
public void columnAdded(TableColumnModelEvent e) {}  
public void columnRemoved(TableColumnModelEvent e) {}  
public void columnMoved(TableColumnModelEvent e) {}  
public void columnMarginChanged(ChangeEvent e) {}
```

(6) 向 `DemoActionEvent` 中添加一个签名为 `private void initialJTable()` 的方法, 并将下列代码添加到该方法中:

```
DealAction da=new DealAction(); //创建监听器对象
jTableMyTable.getSelectionModel().addListSelectionListener(da); //注册
表格模型监听器
jTableMyTable.getColumnModel().addColumnModelListener(da); //注册列模型监听器
```

(7) 将下列代码添加到 DemoActionEvent 类的构造器中:

```
this.initialJTable();
```

(8) 将下列代码添加到 Demo 类的 public static void main(String args[])方法中:

```
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoActionEvent ().setVisible(true);
    }
});
```

(9) 编译并运行程序, 结果如图 12-17 所示。

选中内容为“318320”的单元格, 此时界面如图 12-18 所示。可以看到在窗口的底端出现了“当前选择为第 1 行第 2 列, 值为 318320”的提示信息。

选中“boyngking”单元格, 界面如图 12-19 所示。

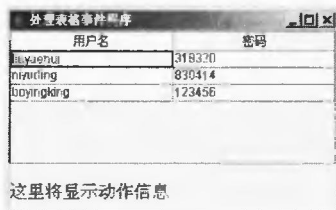


图 12-17 程序运行界面

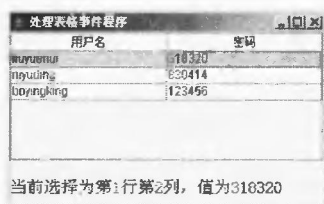


图 12-18 选中“318320”单元格

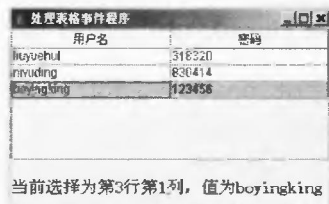


图 12-19 选中“boyngking”单元格

该程序还有其他几种运行状态, 读者可以自己试验这几种情况, 这里不再详细介绍。

12.7 表格编辑器与绘制器

创建使用表格的应用程序不仅可以使使用表格模型与列模型来表示数据, 还可以使用表格编辑器与绘制器控制单元格内容的编辑与表示形式。本节将对表格编辑器与绘制器展开讨论, 主要包括如下内容:

- 表格绘制器简介;
- 表格编辑器简介;
- 在 NetBeans 中开发自定义表格编辑/绘制器的程序实例。

12.7.1 表格绘制器简介

在默认情况下, 表格对每一个单元格中的对象都会调用 toString 方法转换成字符串来显示, 这显然不能完全满足开发的需要。因此, 在应用程序开发中可能常常需要自定义表格的表现形式, 此时会用到表格绘制器。

在 Swing 中, 一般使用实现了 `TableCellRenderer` 接口的类作为表格的绘制器。Swing 提供了一个实现了 `TableCellRenderer` 接口的类 `DefaultTableCellRenderer`, 一般情况下都会使用该类的对象作为表格的绘制器。下列代码介绍了如何使用 `DefaultTableCellRenderer`:

```
DefaultTableCellRenderer myCellRenderer=new DefaultTableCellRenderer();  
myJTable.getColumnModel().getColumn(4).setCellRenderer(myCellRenderer);
```

以上代码创建了 `DefaultTableCellRenderer` 对象 `myCellRenderer` 并将表格的第 4 列的绘制器指定为 `myCellRenderer`。

使用 `DefaultTableCellRenderer` 作为其绘制器的表格, 其单元格内容都使用字符串来表示。如果希望绘制器能够绘制图标和其他内容, 则可以扩展 `DefaultTableCellRenderer`。

如果重写了表格模型中的 `getColumnClass` 方法 (该方法返回描述指定列的类型的类), 则 `JTable` 会根据返回的类的不同为该列选择一种相应的绘制器, 下面列出了每种类型对应的绘制器。

- **Boolean**: 使用复选框来表示;
- **Number**: 使用一个右对齐的标签来表示;
- **Byte/Short/Integer/Long/Double/Float**: 与 `Number` 类型相同, 但是对象到文本的转换是通过 `NumberFormat` 实例来完成;
- **Object**: 使用一个显示字符串值的标签来表示。

要确定在同一列显示的单元格的绘制器, 表格首先要确定是否为该列指定绘制器。如果没有指定, 表格将调用表格模型中能够获得该列单元格数据类型的 `getColumnClass` 方法。然后表格将列数据类型与该表格注册的各个绘制器所支持的数据类型列表进行比较, 从而确定使用哪种绘制器。

提示

上面所列出的几种类型在表格中已经自带了专用的绘制器。如果不是上面所列出的几种除 `Object` 之外的类型, 则要看作 `Object` 来处理。

如果是上面未列出的数据类型, 开发人员可以通过实现 `TableCellRenderer` 接口来为该类型定义绘制器。`TableCellRenderer` 接口只定义了 `getTableCellRendererComponent` 一个方法, 当表格想要绘制单元格时该方法就会被调用。

下列代码通过实现 `TableCellRenderer` 接口定义了自己的表格绘制器。

```
Class ColorCellRenderer implements TableCellRenderer{  
    //重写超类方法  
    public Component getTableCellRendererComponent(JTable table,  
        Object value,boolean isSelected,boolean hasFocus,int row,int column){  
        myPanel.setBackground((Color)value); //自定义绘制器代码  
        return myPanel;  
    }  
    private JPanel myPanel=new JPanel();  
}
```

该绘制器用来显示类型为 `Color` 的列, 只返回一个指定颜色的面板, 其背景色通过 `value` 参数来传递。接下来只需要告诉表格使用该绘制器绘制类型为 `Color` 的列, 代码如下:

```
myJTable.setDefaultRenderer(Color.class,new ColorCellRenderer() );
```

上面代码的含义是告诉表格使用自定义的 `ColorCellRenderer` 绘制器来绘制类型为 `Color` 列的单元格。

12.7.2 表格编辑器简介

除了自定义表格绘制器以外，还可以自定义表格编辑器。在 `Swing` 中，一般使用实现了 `TableCellEditor` 接口的类作为表格的编辑器。

`TableCellEditor` 接口是定义任何表格编辑器的起点，定置单元格编辑器需要实现该接口。该接口的操作比较复杂，从 `JDK1.3` 开始 `Swing` 提供了一个 `AbstractCellEditor` 抽象类，可以通过扩展该类来定置表格编辑器，这样简单一些。

`TableCellEditor` 接口的 `getTableCellEditorComponent` 方法用来请求一个控件，以便编辑单元格。它与 `TableCellRenderer` 接口的 `getTableCellRendererComponent` 方法很相似，只不过没有 `hasFocus` 参数。由于单元格被编辑了，因此可以认为其拥有了焦点。当编辑正在进行时，编辑器将临时取代绘制器进行显示。

如果需要扩展 `AbstractCellEditor` 来定置自己的编辑器，可以按如下步骤进行。

- (1) 扩展 `AbstractCellEditor` 类并且实现 `TableCellEditor` 接口。
- (2) 重写 `getTableCellEditorComponent` 方法以便提供一个控件。



说明

该控件可以是可编辑的控件，如文本框；还可以是不可编辑的控件，如单击该控件可弹出对话框。

(3) 定义 `shouldSelectCell`、`stopCellEditing`、`cancelCellEditing` 方法，用于控制单元格的开始编辑、完成编辑和取消编辑。



说明

`stopCellEditing` 与 `cancelCellEditing` 方法需要调用超类方法，以确保监听器能够得到这些通知。

(4) 定义 `getCellEditorValue` 方法以便返回作为单元格的值。

(5) 最后，还要调用 `stopCellEditing` 与 `cancelCellEditing` 方法，指明用户何时完成操作。

`Swing` 提供了一个扩展了 `AbstractCellEditor` 的类 `DefaultCellEditor`，使用它可以很方便的创建使用文件字段、复选框和组合框的编辑器。下面的代码创建了一个使用组合框的编辑器。

```
TableColumn myColumn=table.getColumnModel().getColumn(2); //获取表格的第2列
.....
JComboBox comboBox=new JComboBox();
comboBox.add("sing");
comboBox.add("dance");
comboBox.add("football");
comboBox.add("basketball");
myColumn.setCellEditor(new DefaultCellEditor(comboBox) );
```

`DefaultCellEditor` 提供了 3 个构造器，表 12-19 列出了这 3 个构造器及其说明。

`DefaultCellEditor` 类提供了一些方法供开发人员使用，表 12-20 给出了常用的方法及其说明。

表 12-19 DefaultCellEditor 类构造器及说明

构造器	说 明
DefaultCellEditor(JTextField textField)	构造一个使用文本字段的 DefaultCellEditor 对象
DefaultCellEditor(JCheckBox checkBox)	构造一个使用复选框的 DefaultCellEditor 对象
DefaultCellEditor(JComboBox comboBox)	构造一个使用组合框的 DefaultCellEditor 对象

表 12-20 DefaultCellEditor 常用方法及说明

方 法	说 明
getComponent()	返回对编辑器组件的引用
setClickCountToStart(int count)	指定开始编辑所需的单击次数
getClickCountToStart()	返回开始编辑所需的单击次数
getCellEditorValue()	将消息从 CellEditor 转发到 delegate
isCellEditable(EventObject anEvent)	将消息从 CellEditor 转发到 delegate
shouldSelectCell(EventObject anEvent)	将消息从 CellEditor 转发到 delegate
stopCellEditing()	将消息从 CellEditor 转发到 delegate
cancelCellEditing()	将消息从 CellEditor 转发到 delegate
getTableCellEditorComponent(JTable table, Object value, boolean isSelected, int row, int column)	实现 TableCellEditor 接口

DefaultCellEditor 还可以使用父类中的一些方法进行相应的操作,需要用到的用户可以查阅相关的 API,这里不再详细介绍。

12.7.3 在 NetBeans 中开发自定义表格编辑/绘制器的程序

前面两个小节对表格的编辑器与绘制器进行了简单的介绍,本节将通过一个例子介绍如何在 NetBeans 中开发使用自定义编辑器与绘制器的程序。

打开前面创建的项目 table,按照如下步骤完成该例。

(1) 向 table 中添加通过 JFrame Form 模版创建的类 DemoUseRenderAddEditor。

(2) 向 DemoUseRenderAddEditor 中添加一个 JTable,将该 JTable 的名称修改为 jTableMyTable。在设置模型对话框中设置其模型,在“User Code”文本框中添加如下代码:

```
new MyTableModel();
```


(3) 向 DemoUseRenderAddEditor 中添加一个名称为 MyTableModel,并扩展了 AbstractTableModel 类的内部类。然后将下列代码添加到该类的类中:

```
private String[] columnNames = { "姓名",
                                "喜欢的颜色",
                                "爱好",
                                "年龄",
                                "从事该职业"};

private Object[][] data = {
    {"王政", new Color(153, 0, 153),
     "玩耍", new Integer(2), new Boolean(false)},
    {"伊飞", new Color(51, 51, 153),
```



```
        "上课", new Integer(22), new Boolean(true)),
        {"王烨", new Color(51, 102, 51),
        "聊天", new Integer(2), new Boolean(false)},
        {"阿森", Color.red,
        "写程序", new Integer(19), new Boolean(true)},
        {"小伟", Color.pink,
        "玩扑克", new Integer(24), new Boolean(false)}
    };
    //重写 getColumnCount 方法
    public int getColumnCount() {
        return columnNames.length;
    }
    //重写 getRowCount 方法
    public int getRowCount() {
        return data.length;
    }
    //重写 getColumnName 方法
    public String getColumnName(int col) {
        return columnNames[col];
    }
    //重写 getValueAt 方法
    public Object getValueAt(int row, int col) {
        return data[row][col];
    }
    /*
    *表格使用该方法决定默认的单元格的编辑器。
    *如果不重写该方法，则最后一列内容显示为“true”或者“false”的字符串，
    *而不是用复选框显示
    */
    public Class getColumnClass(int c) {
        return getValueAt(0, c).getClass();
    }
    //设置单元格的可编辑性
    public boolean isCellEditable(int row, int col) {
        if (col < 1) {
            return false;
        } else {
            return true;
        }
    }
    public void setValueAt(Object value, int row, int col) {
        data[row][col] = value;
        fireTableCellUpdated(row, col); //通知监听器单元格内容已经改变
    }
}
```

 说明 该类用来初始化表格模型。

(4) 向 DemoUseRenderAddEditor 中添加一个名称为 ColorRenderer、扩展了 JLabel 并实现了 TableCellRenderer 接口的内部类。将下列代码添加到该类中:

```
public ColorRenderer() {
    setOpaque(true); //该方法用来显示背景颜色
}
//重写父类方法
public Component getTableCellRendererComponent(
    JTable table, Object color,
    boolean isSelected, boolean hasFocus,
    int row, int column) {
    Color newColor = (Color)color;
    setBackground(newColor);
    return this; //返回该类的实例
}
```

说明 该类用来定义表格的绘制器。

(5) 向 DemoUseRenderAddEditor 中添加一个名称为 ColorEditor 的内部类。它扩展了 AbstractCellEditor 类, 实现了 TableCellEditor 与 ActionListener 接口, 定义了表格的编辑器。将下列代码添加到该类中:

```
Color currentColor;
JButton button;
JColorChooser colorChooser;
JDialog dialog;
protected static final String EDIT = "edit";
public ColorEditor() {
    button = new JButton();
    button.setActionCommand(EDIT);
    button.addActionListener(this);
    //创建颜色选择器对象, 并使用该颜色选择器创建对话框
    colorChooser = new JColorChooser();
    dialog = JColorChooser.createDialog(button, "选择颜色", true,
    colorChooser, this, null);
}
//该方法根据用户的选择确定背景颜色
public void actionPerformed(ActionEvent e) {
    if (EDIT.equals(e.getActionCommand())) {
        button.setBackground(currentColor);
        colorChooser.setColor(currentColor);
        dialog.setVisible(true);
        fireEditingStopped(); //通知绘制器编辑完成
    } else { //用户按下了“确定”按钮
        currentColor = colorChooser.getColor();
    }
}
```

```

    }
    //实现 getCellEditorValue 方法
    public Object getCellEditorValue() {
        return currentColor;
    }
    //实现 getTableCellEditorComponent 方法
    public Component getTableCellEditorComponent(JTable table,
        Object value,boolean isSelected,int row,int column) {
        currentColor = (Color)value;
        return button;
    }
}

```

(6) 向 DemoUseRenderAddEditor 中添加一个签名为 `private void initialJTable()` 的方法, 并将下列代码添加到该方法中:

```

jTableMyTable.setDefaultRenderer(Color.class,new ColorRenderer());
jTableMyTable.setDefaultEditor(Color.class,new ColorEditor());

```

- 第一行设置 ColorRenderer 为 Color 类型的绘制器。
- 第二行设置 ColorEditor 为 Color 类型的编辑器。

(7) 向 DemoUseRenderAddEditor 类的构造器中添加如下代码:

```
this.initialJTable();
```

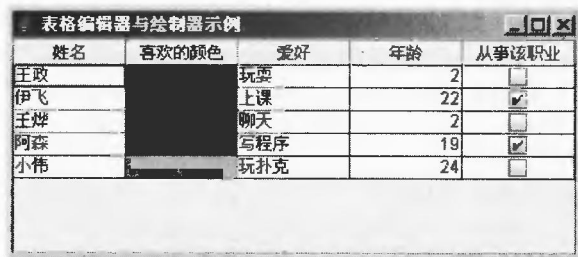
(8) 将下列代码添加到 Demo 类的 `public static void main(String args[])` 方法中:

```

java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new DemoUseRenderAddEditor().setVisible(true);
    }
});

```

(9) 编译并运行程序, 结果如图 12-20 所示。



姓名	喜欢的颜色	爱好	年龄	从事该职业
王政		玩票	2	<input type="checkbox"/>
伊飞		上课	22	<input checked="" type="checkbox"/>
王烨		聊天	2	<input type="checkbox"/>
阿森		写程序	19	<input checked="" type="checkbox"/>
小伟		玩扑克	24	<input type="checkbox"/>

图 12-20 程序运行结果

(10) 单击“王政”右侧的“喜欢的颜色”单元格, 将弹出选择颜色对话框, 如图 12-21 所示。

选中一定的颜色后单击“确定”按钮, 则设置的单元格将会变为相应的颜色, 如图 12-22 所示。本程序还有其他运行状态, 这里就不全部给出了, 有兴趣的读者可以试验这些情况。

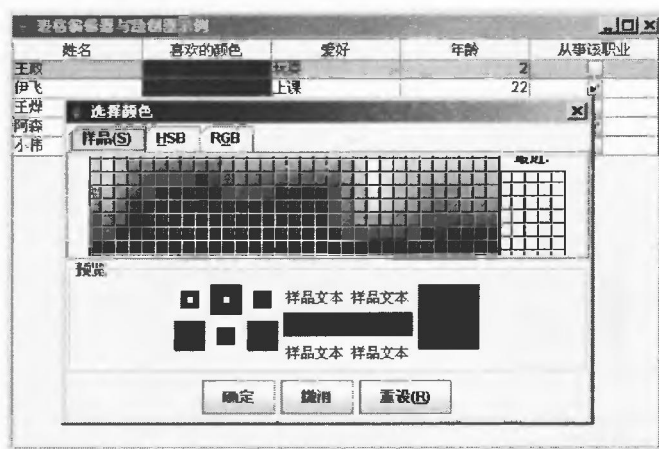


图 12-21 选择单元格颜色窗口

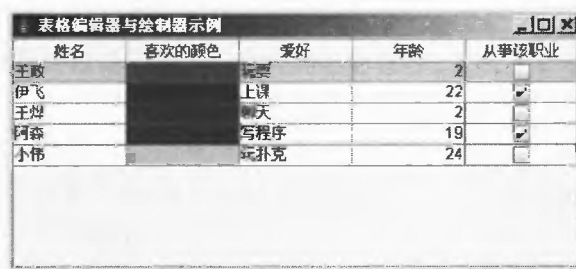


图 12-22 设置颜色后的窗口

12.8 小结

本章通过具体的实例，讲述了如何使用 Swing 提供的各种表格控件，包括 JTable、表格模型、表格列、表格列模型、表格的编辑器与绘制器等内容。可以看到，在 NetBeans 中开发使用表格的程序非常方便，只需要通过鼠标的拖曳即可完成大部分界面的开发，而不需要手工编写太多的代码。

第 13 章

Swing 线程

不正确的 Swing 线程使用是造成 Swing 程序运行缓慢、无响应和不稳定的主要原因之一。开发人员对 Swing 线程模型的误解以及不能够保证线程的正确执行都能造成 Swing 线程不正确。本章将对与 Swing 线程有关的内容展开讨论，主要包括如下内容：

- ✎ 一个存在 BUG 的简单程序；
- ✎ Swing 线程的工作原理；
- ✎ 解决 Swing 多线程问题。

13.1 一个存在 BUG 的简单程序

当开发具有图形用户界面的时候，可能会经常出现用户界面不灵敏的现象。用户界面不灵敏是指：

- ✎ 不响应事件；
- ✎ 使用代码更新用户界面，但实际上用户界面并没有更新。

本节主要讲解一个用户界面不灵敏的简单 Swing 程序，其完整代码（BugExample.java）如下：

```
package org.netbeans.swing.thread;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class BugExample extends JFrame implements ActionListener{
    JTextField myJTextField;
    JButton countButton;
    public BugExample(){
        //设置布局管理器
        this.setLayout(null);
        myJTextField=new JTextField();
```

```
countButton=new JButton("运算");
//设置控件的大小以及位置
myJTextField.setBounds(20,10,150,30);
countButton.setBounds(180,10,60,30);
//为 countButton 添加监听器
countButton.addActionListener(this);
this.add(myJTextField);
this.add(countButton);
//设置窗口的大小、位置、图标以及可见性
this.setBounds(250,250,270,100);
this.setTitle("存在 BUG 的应用程序");
this.setVisible(true);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
//处理用户的动作事件
public void actionPerformed(ActionEvent e){
    int number=0;
    //如果用户按下了 countButton 按钮
    if(e.getSource()==countButton){
        //循环输出 number 的值
        while(number<100000){
            System.out.println(number++);
        }
        //设置 myJTextField 控件的内容
        this.myJTextField.setText(""+number);
    }
}
public static void main(String args[]){
    new BugExample();
}
}
```

编译并运行程序，结果如图 13-1 所示。单击运算按钮，结果如图 13-2 所示。图 13-2 所示窗口的状态会保持一段时间，此时在控制台中会逐个输出 0~99999 之间的数字。输出完毕后，结果如图 13-3 所示。



图 13-1 程序运行结果



图 13-2 单击“运算”按钮后结果

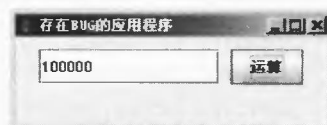


图 13-3 程序运行完毕后的结果

从图 13-2 中可以看到，释放按钮以后，该按钮仍呈现被按下的状态。这说明，虽然 `actionPerformed` 方法通知了按钮绘制为非按下外观，但是该方法没有被返回。

产生用户界面不灵敏的现象，很大程度上都与线程有关。在后面的章节中，将对 Swing 的线程展开详细的讨论。

13.2 Swing 线程的工作原理

上一节中给出了一个存在 bug 的简单程序，本节将对 Swing 线程的工作原理进行简单介绍，剖析产生上节所述问题的原因，主要包括如下内容：

- 事件分发线程工作原理；
- 事件分发线程模型引发的问题。

13.2.1 事件分发线程工作原理

对于 Java 的初学者而言，在没有自己编写多线程程序时总觉得编写的程序是单线程的。其实不然，可以说几乎所有的 Java 程序都是多线程的。因为 Java 中不但有开发人员自己开发的线程，而且还有很多后台系统线程。

事件分发线程就是后台系统线程的一种，在 Swing 界面工作的过程中，对所有事件的处理都是在事件分发线程中执行的。下面的示意图 13-4 简单说明了事件分发线程的工作原理。



图 13-4 事件分发线程的工作原理

- 途中由左至右表示时间流逝的方向，灰色的部分表示事件分发线程处于等待状态，当前没有任务执行。
- 黑色的“按钮 1”、“按钮 2”及“按钮 1”部分表示事件分发线程正在执行对相应按钮事件处理的代码。

从上面的途中可以看出，如果事件分发线程正在执行事件处理代码的同时，又有新的事件触发，就有可能产生界面迟滞，不响应的情况。

13.2.2 事件分发线程模型引发的问题

通过 13.2.1 小节的介绍可以看到，事件分发线程是与 Swing 控件进行交互处理事件的惟一的线程。其不但要进行事件处理，还有很多其它职责，例如对图形界面的绘制等。下面就介绍一下容易引发问题的位置以及线程安全的方法：

1. 容易引发问题的位置

容易引发问题的位置往往在开发人员自己编写的事件处理代码中。从前面的介绍中已经知道，事件分发线程的任务是很繁重的，而事件处理方法又要在事件分发线程中执行。这时，如果在事件处理方法中有过多的、占用大量时间的任务就会产生问题。

比如在上节给出的程序中，用来处理按钮事件的 `actionPerformed()` 方法，在事件分发线程中执行，则其中的输出操作也将在事件分发线程中执行。这大量占用了事件分发线程正常的工作时间，阻止它处理其他任务，如绘制控件、响应鼠标移动、处理按钮事件等。

这样，用户以为应用“死掉了”，但实际上并不是这样。

2. 线程安全的方法

在 Swing 中有这样的规则：一旦 Swing 组件被具现化（realized），所有可能影响或依赖于组

件状态的代码都应该在事件分发线程中执行。例如：用来构造用户界面的各种方法就应该在界面具现化之前或事件分发线程中被调用。如果违反了这个规则，代码的执行可能会带来问题。

说明：具现化的意思是组建的 `paint()` 方法已经或可能会被调用。

在这里，线程安全的方法是指在具现化以后，不在事件分发线程中调用的，影响或依赖于组件状态的方法。这些方法主要有：`repaint()`、`revalidate()`、和 `invalidate()`。

- `repaint()` 和 `revalidate()` 方法为事件派发线程对请求排队，并分别调用 `paint()` 和 `validate()` 方法。
- `invalidate()` 方法只在需要确认时标记一个组件和它的所有直接祖先。

提示 一个 `applet` 的图形用户界面可以在 `init()` 方法中构造，因为现有的浏览器都不会在一个 `applet` 的 `init()` 和 `start()` 方法被调用前绘制它。因而在一个 `applet` 的 `init()` 方法中构造图形用户界面是安全的，只要不对 `applet` 中的对象调用 `show()` 或 `setVisible(true)` 方法。

13.3 解决 Swing 单线程问题

可以使用 `SwingUtilities` 类提供的方法来解决 Swing 的单线程问题。`SwingUtilities` 类提供了两个方法来帮助开发人员在事件分发线程中执行代码：

- `invokeLater (Runnable doRun)`：要求在事件分发线程中执行某些代码。这个方法会立即返回，不会等待代码执行完毕。
- `invokeAndWait (Runnable doRun)`：行为与 `invokeLater` 类似，只不过这个方法会等待代码执行完毕。

可以从任何线程调用 `invokeLater` 方法以请求事件分发线程运行特定代码。必须把要运行的代码放到一个 `Runnable` 对象的 `run` 方法中，并将此 `Runnable` 对象设为 `invokeLater` 的参数。`invokeLater` 方法会立即返回，不等待事件分发线程执行指定代码。下面是一个使用 `inookeLater` 方法的例子：

```
Runnable doWorkRunnable = new Runnable() {
    public void run() {
        //业务代码
    }
};
SwingUtilities.invokeLater(doWorkRunnable);
```

`invokeAndWait` 方法与 `invokeLater` 方法不同，`invokeLater` 方法采用的是异步调用，而 `invokeAndWait` 方法采用的是同步调用，其会等事件分发线程执行完指定的方法才返回。

下面是一个使用 `invokeAndWait` 方法的例子：

```
Runnable showModalDialog = new Runnable() {
    public void run() {
        JOptionPane.showMessageDialog(myMainFrame, "Hello There");
    }
};
SwingUtilities.invokeAndWait(showModalDialog);
}
```


一般情况下,开发人员应该尽量使用 `invokeLater` 方法,而尽量少使用 `invokeAndWait` 方法。**注意** 如果必须要使用 `invokeAndWait` 方法,一定要确保调用 `invokeAndWait` 方法的线程不会在 `invokeAndWait` 方法返回之前持有任何其他线程可能需要的对象锁。

还可以编写自己的线程,在其中执行业务逻辑而不占用分发线程。下面使用该方法改造了本章第一节给出的程序(`NoBugExample.java`)。

```
package org.netbeans.swing.thread;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class NoBugExample extends JFrame implements ActionListener{
    JTextField myJTextField;
    JButton countButton;
    public NoBugExample(){
        //设置布局管理器
        this.setLayout(null);
        myJTextField=new JTextField();
        countButton=new JButton("运算");
        //设置控件的大小以及位置
        myJTextField.setBounds(20,10,150,30);
        countButton.setBounds(180,10,60,30);
        //为 countButton 添加监听器
        countButton.addActionListener(this);
        this.add(myJTextField);
        this.add(countButton);
        //设置窗口的大小、位置、图标以及可见性
        this.setBounds(250,250,270,100);
        this.setTitle("改造后的应用程序");
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    //处理用户的动作事件
    public void actionPerformed(ActionEvent e) {
        final int number=100000;
        //如果用户按下了 countButton 按钮
        if(e.getSource()==countButton){
            //创建 Thread 类的对象,其为匿名内部类
            Thread mythread=new Thread(){
                public void run(){
                    //循环输出 number 的值
                    for(int i=0;i<number;i++){
                        System.out.println(i);
                    }
                }
            };
            //启动线程 mythread
        }
    }
}
```

```
        mythread.start();  
        //设置 myJTextField 的内容  
        this.myJTextField.setText(""+number);  
    }  
}  
public static void main(String args[]){  
    new NoBugExample();  
}  
}
```

编译并运行程序,结果与 13-1 所示一致。单击运算按钮后,结果如图 13-5 所示。

可以看出,改造后的程序使用自定义的线程执行业务逻辑,不使用分发线程执行业务逻辑。分发线程可以很迅速地对按钮的外观进行重绘并响应其他事件,不需要等待业务逻辑返回后再重绘按钮的外观、响应其他事件。



图 13-5 改造后程序运算界面

13.4 小结

本章主要讨论了与 Swing 的线程有关的内容,介绍了一个用户界面不灵敏的程序,并就此对 Swing 的工作原理进行了详细的讨论,另外给出了解决 Swing 单线程的方法并改造了本章开始给出的用户界面不灵敏的程序。

本篇为 Web 开发篇，其中结合 Tomcat 详细地讲述了如何在 NetBeans 中进行 Web 程序的开发。本篇主要包括如下内容：

- 安装与配置 Tomcat;
- JSP 技术介绍及开发使用 JSP 的程序;
- Servlet 介绍及开发使用 Servlet 的程序;
- JavaBean 介绍及开发使用 JavaBean 的程序;
- 开发具有 MVC 架构的网上商店。

第 3 篇 Web 开发篇

第 14 章 Tomcat 配置及应用

第 15 章 JSP 技术及应用

第 16 章 Servlet 技术及应用

第 17 章 JavaBean 组件模型

第 18 章 开发 MVC 架构的网上商店

第 14 章

Tomcat 配置及应用

对于开发好的 Web 应用，需要部署到一定的 Web 容器中才能运行。Tomcat 是开发 Web 应用程序时经常用到的 JSP/Servlet 容器。本章将对与 Tomcat 有关的内容进行详细的讲述，主要包括如下内容：

- Tomcat 简介；
- Tomcat 的安装与测试；
- 向 NetBeans 中添加 Tomcat；
- 在 NetBeans 中配置 Tomcat；
- 其他 Web 服务器简介。

14.1 Tomcat 简介

Tomcat 是一个开源的 JSP/Servlet 容器，也是当前最为流行的 JSP/Servlet 容器。本节将会对 Tomcat 进行简单的介绍，主要包括如下内容：

- Web 程序介绍；
- Tomcat 与传统的 Web 服务器；
- Tomcat 的 Realm 支持；
- Tomcat 与 J2EE 服务器。

14.1.1 Web 程序介绍

与传统的 Java Desktop 应用程序不同，Java Web 应用程序一般情况下是打包成 WAR(Web Archive) 文件并且不能直接执行该程序，而是要将该程序部署在对应的 Web 容器中（如：Tomcat），通过浏览器访问该程序。

WAR 也是 Sun 提出的一种 Web 应用程序组织标准，与常用的 JAR 文件类似，WAR 文件也是包含许多文件的一个压缩包。

WAR 包中的文件按一定目录结构来组织，下面给出了 WAR 包中目录的组织方式。

- WAR 包的根目录下包含有 Html 和 JSP 文件或者包含这两种文件的目录;
- WAR 包根目录下包含有一个 WEB-INF 目录;
- WEB-INF 目录下包含有一个 web.xml 文件,此文件是该 Web 应用程序的配置文件;
- WEB-INF 目录下包含有一个 classes 目录,此目录下包含编译好的 Servlet 类和 Servlet 所依赖的其他类如 JavaBean;
- WEB-INF 目录下包含有一个 lib 目录,此目录下包含 web 应用程序中用到的其他 Java 类的 jar 包。

❗注意 如果不需要, WEB-INF 下可以没有 classes 与 lib 目录。

图 14-1 给出了 WAR 包的组织结构图。

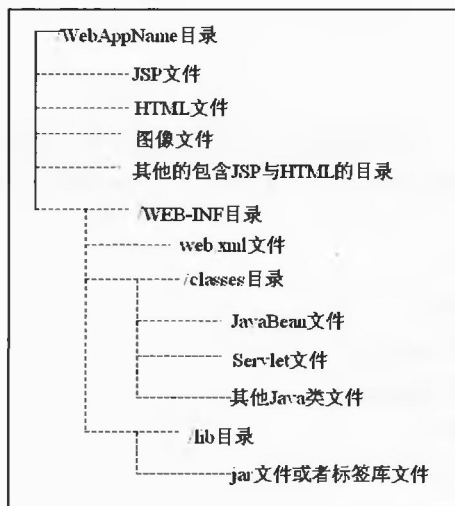


图 14-1 WAR 包组织结构图

在 Tomcat 中部署 web 应用程序非常简单: 只需将已经打包的 WAR 文件放到 Tomcat 的 webapp 目录下。当访问该 web 应用程序时, Tomcat 会自动检测到这个文件, 并将其解压。

默认情况下, Tomcat 也提供了一个 web 应用程序——manager, 访问这个应用需要用
❗提示 用户名和密码, 用户名和密码存储在 xml 文件中。通过该 web 应用程序可以远程部署和删除 Web 应用程序。

14.1.2 Tomcat 与传统的 Web 服务器

Tomcat 不仅仅是一个 JSP/Servlet 容器, 还具有传统的 Web 服务器功能。Tomcat 处理静态 Html 的能力不如 Apache, 但可以将 Tomcat 和 Apache 集成到一起使用, 以提高服务器的使用效率。一般的使用方法如下:

- 用 Apache 处理静态 Html;
- 用 Tomcat 处理 JSP 和 Servlet。

❗提示 如果需要将 Tomcat 和 Apache 集成到一起使用, 需要修改 Apache 和 Tomcat 的配置文件。

在 Tomcat 中, 可以利用 Servlet 提供的事件监听器功能来对 Web 应用或者 Session 实行监听。除此之外, Tomcat 还提供了以下显著的功能:

- 与 SSL 集成到一起, 实现加密传输;
- 提供对 JNDI 的支持。

14.1.3 Tomcat 的 Realm 支持

Realm 与 Oracle 系统中经常使用的 role 非常相似。在 Oracle 系统中, 一个 role 对应着对一组特定资源的访问权限, 具有特定 role 的用户只能访问 role 中指定的资源。Tomcat 使用 Realm 使某些特定的用户组具有访问某些特定的 Web 应用的权限, 而没有权限的用户不能访问这个应用。

Tomcat 提供了 3 种不同的 Realm 对访问某个 Web 应用程序的用户进行相应的验证。

- JDBCRealm: 此种类型的 Realm 将用户信息存储在数据库里, 通过 JDBC 从数据库中提取用户信息并进行验证。
- JNDIRealm: 此种类型的 Realm 将用户信息存储在基于 LDAP 等目录服务的服务器里, 通过 JNDI 技术从 LDAP 服务器中获取用户信息并进行验证。
- MemoryRealm: 此种类型的 Realm 将用户信息存储在 xml 文件中, 对用户进行验证时, 将会从相应的 xml 文件中提取用户信息。上面提到的 manager 应用在进行验证时即使用此种 Realm。

14.1.4 Tomcat 与 J2EE 服务器

通常所说的 J2EE 应用服务器 (如 WebLogic) 与 Tomcat 是有一定区别的。J2EE 应用服务器提供更多的 J2EE 特性, 如支持 EJB、JMS、JAAS 等, 同时也支持 JSP 和 Servlet。而 Tomcat 的功能则没有那么强大, 如不提供对 EJB 的支持。

但是在实际的应用中, 很多的中小应用可能不需要采用 EJB 等复杂技术, 仅仅使用 JSP 和 Servlet 就已经足够了。这时如果使用 J2EE 应用服务器就有些浪费了, 而 Tomcat 短小精悍, 配置方便, 又能满足应用需求, 这种情况下自然会选择 Tomcat。

另外还可以将 Tomcat 与 J2EE 应用服务器一起使用以提高容器的使用效率。一般的使用方法如下:

- 用 Tomcat 处理 Web 应用;
- 用 J2EE 应用服务器处理企业级应用。

除此之外, Tomcat 也可以与其他一些软件集成起来实现更多的功能, 如与 Cocoon (Apache 的另外一个项目) 集成起来开发基于 Xml 的应用, 与 OpenJMS 集成起来开发基于 JMS 的应用等。

提示 基于 Tomcat 的开发其实主要是 JSP 与 Servlet 的开发, 在本章后面的几节中会详细的讨论如何开发 JSP 与 Servlet。

14.2 安装与配置 Tomcat

在使用 Tomcat 之前首先要安装与配置 Tomcat。本节将为读者讲述如何安装与配置 Tomcat。主要包括如下内容:

- 安装 Tomcat;
- 测试 Tomcat 是否可用;
- 开发并部署第一个 JSP。

14.2.1 安装 Tomcat

安装 Tomcat 之前,首先要获取 Tomcat 的安装文件。可以从以下地址下载 Tomcat 的安装文件: <http://jakarta.apache.org/tomcat>。获取 Tomcat 的安装文件以后,按照如下步骤进行安装。

(1) 双击下载后的可执行文件,经过一段时间的初始化工作以后,出现如图 14-2 所示窗口。

(2) 单击“Next”按钮,进入许可协议界面,如图 14-3 所示。这里列出了使用 Tomcat 必须要遵守的一些协议,用户只有接受这些协议才能够继续安装。



图 14-2 安装 Tomcat 的欢迎界面

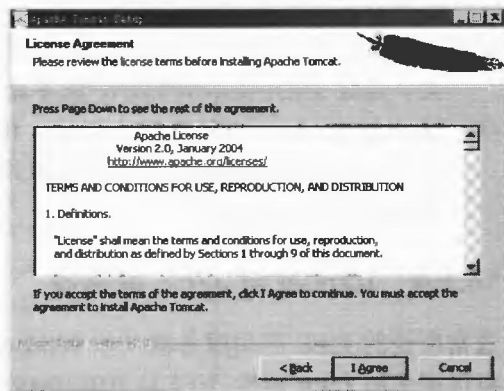


图 14-3 接受许可协议界面

(3) 单击“I Agree”按钮,选择安装的内容,如图 14-4 所示。

图 14-4 列出了可以选择安装的类型以及要安装的组件。可供选择的安装类型有 4 种: **说明** Custom (自定义)、Normal (普通)、Full (完全安装) 以及 Minmun (最小安装),在此选择完全安装选项。其余选项均使用默认值。

(4) 单击“Next”按钮继续安装,设置安装路径,如图 14-5 所示。

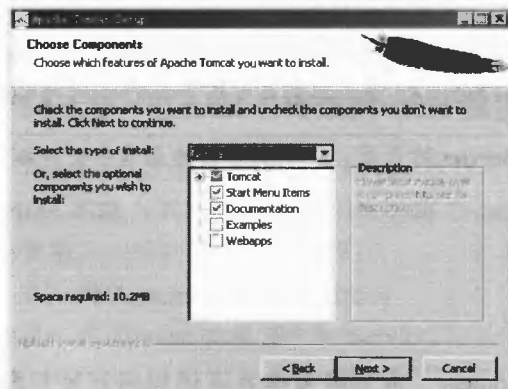


图 14-4 选择要安装的组件

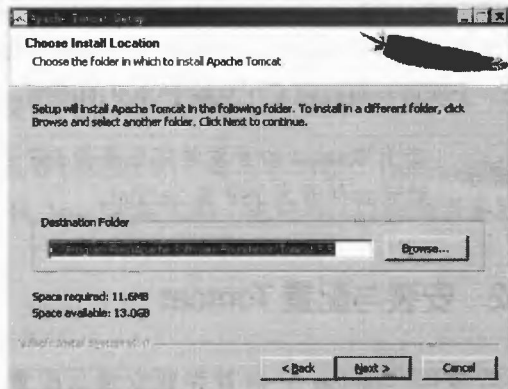


图 14-5 设置安装路径选项

(5) 设置完成以后, 单击“Next”按钮设置端口号码以及管理员登录账号, 如图 14-6 所示。这里设置 Tomcat 的端口号码为 8877。

注意 1024 以下的端口号码, 可能已经被某些特定的系统服务所占用, 因此读者在设置端口号码的时候, 最好不要使用 1024 以下的端口号码。另外, 必须要记清楚该用户名称、密码, 在以后管理 Tomcat 时会用到用户名称、密码。

(6) 单击“Next”按钮继续安装, 设置 JDK 路径 (如 F:\Program Files\Java\jdk1.5.0_06), 如图 14-7 所示。

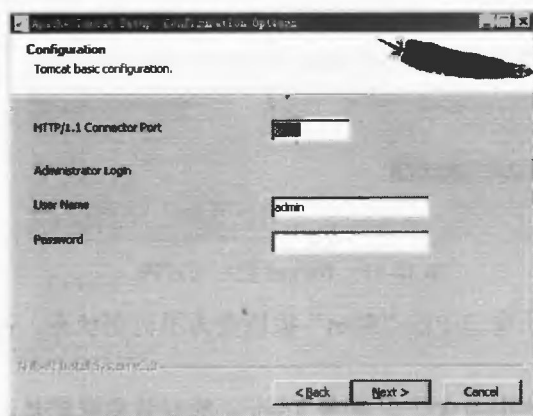


图 14-6 设置端口号码用户名以及密码

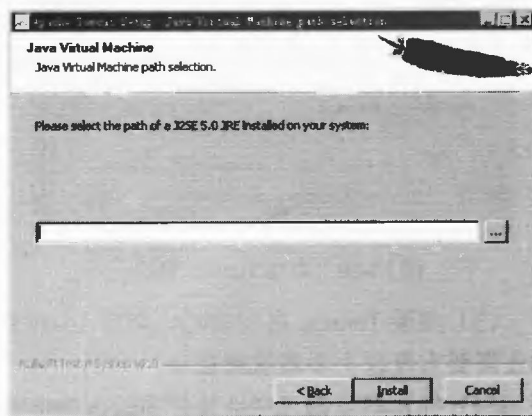


图 14-7 选择 JDK 窗口

(7) 设置完成以后, 单击“Install”按钮开始安装, 如图 14-8 所示。

(8) 该安装过程会持续一段时间, 安装完成以后, 出现如图 14-9 所示窗口。

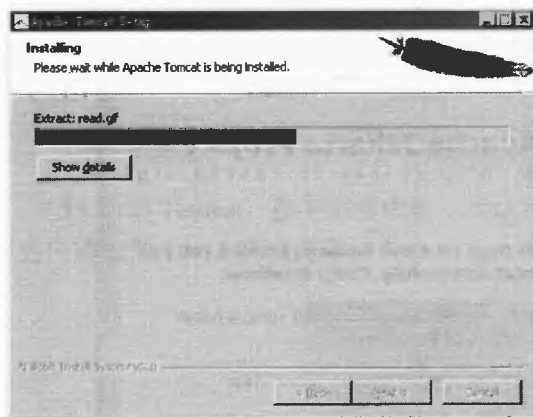


图 14-8 安装 Tomcat 进度窗口



图 14-9 完成 Tomcat 的安装

(9) 将“Run Apache Tomcat”与“Show Readme”选项均设置为非选中状态, 单击“Finish”按钮, 即完成了 Tomcat 的安装。

14.2.2 测试 Tomcat 是否可用

安装完 Tomcat 以后, 需要测试该 Tomcat 是否可以使用。按照以下步骤完成测试过程。

(1) 单击“开始”按钮, 依次选择“程序”/“Apach Tomcat 5.5”/“Configure Tomcat”

选项, 出现如图 14-10 所示窗口。

(2) 单击“Start”按钮启动 Tomcat, 启动过程如图 14-11 所示。

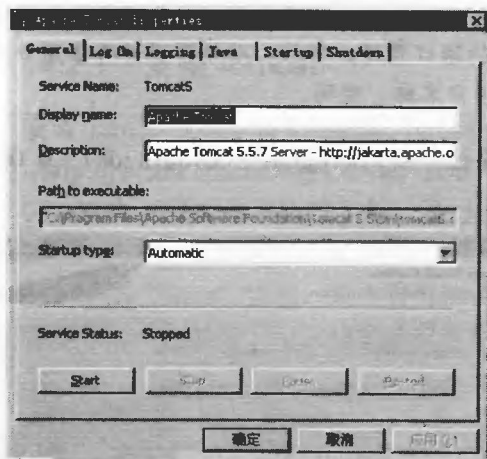


图 14-10 启动 Tomcat 窗口

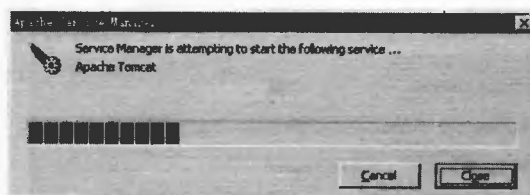


图 14-11 Tomcat 的启动过程

(3) 如果 Tomcat 启动成功, 则图 14-10 所示窗口中的“Start”按钮变为不可用状态。否则还需要重复以上步骤重新启动。

(4) 打开浏览器, 在地址栏中输入 <http://localhost:8877/>, 如果 Tomcat 的安装和配置是正确的, 则在浏览器中可以看到图 14-12 所示界面。

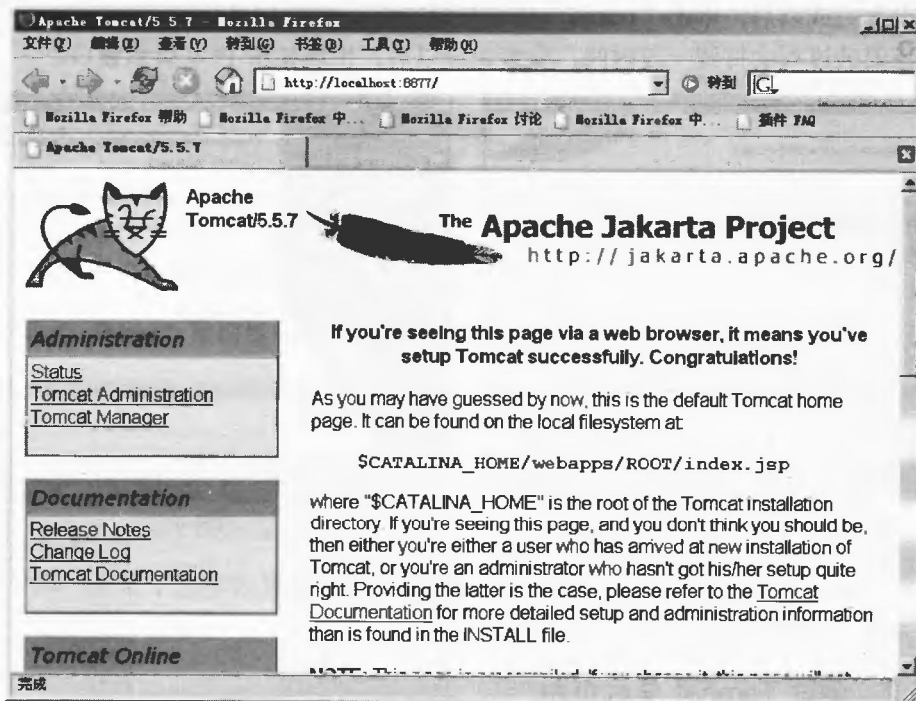


图 14-12 测试 Tomcat 安装是否成功

(5) 如果不能看到图 14-12 所示界面, 说明安装的 Tomcat 存在问题, 如 Tomcat 使用的

JDK 设置不正确等。

14.2.3 开发并部署第一个 JSP

本节将通过一个例子向读者介绍如何在 Tomcat 中部署 JSP 程序。步骤如下：

(1) 在 Tomcat 安装目录下的 webapps 目录下新建一个名称为 14tomcat 的目录，并向其中添加一个名称为 TestTomcat.jsp 的文件，下面给出了 TestTomcat.jsp 的完整代码。

```
<%@ page contentType="text/html; charset=GBK" %>
<html>
<head>
  <title>在 Tomcat 中部署 JSP</title>
</head>
<h1>
<body>
  当前系统时间是: <br>
  <%=new java.util.Date().toLocaleString()%>
</body>
<h1>
</html>
```

(2) 在 14tomcat 目录中新建一个名称为 WEB-INF 的目录，并向其中添加一个名称为 web.xml 的文件，web.xml 文件中内容如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
</web-app>
```

(3) 启动 Tomcat，在浏览器中输入 <http://localhost:8877/14tomcat/TestTomcat.jsp>，浏览器中会出现如图 14-13 所示内容。

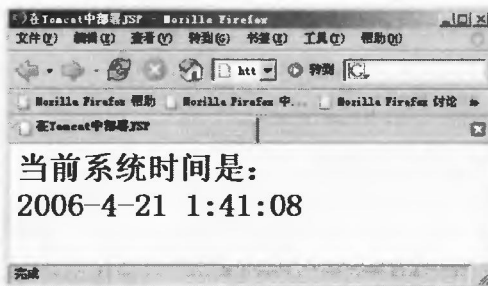


图 14-13 在浏览器中请求 TestTomcat.jsp 页面



提示 web.xml 的内容比较复杂，一般情况下很少自己编写，而是采取拷贝后修改的方法。这样可以避免不必要的错误。

14.3 在 NetBeans 中使用 Tomcat

本节将为读者讲述如何在 NetBeans 中使用 Tomcat，主要包括如下内容：

- 向 NetBeans 中添加 Tomcat；
- 配置 Tomcat；
- 操作 Tomcat；
- 设置 NetBeans 使用的默认浏览器。

14.3.1 向 NetBeans 中添加 Tomcat

安装 NetBeans 时，默认情况下会自动安装 Tomcat。可以在“Runtime”窗口中查看已经安装的 Tomcat，如图 14-14 所示。从图 14-14 中可以看到，默认情况下安装的 Tomcat 版本为 5.5.9。另外，可以按照如下步骤向 NetBeans 中添加操作系统中已经安装的 Tomcat。

(1) 在运行环境 (Runtime) 窗口中右键单击“Server”节点，在弹出的菜单中选择“Add Server”选项来添加 Tomcat 容器，如图 14-15 所示。

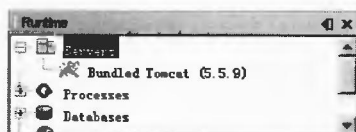


图 14-14 默认安装的 Tomcat

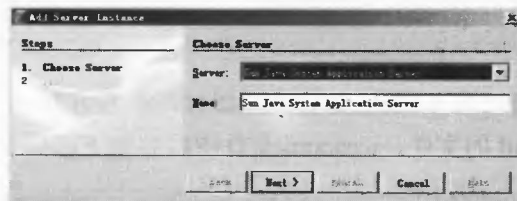


图 14-15 添加 Tomcat 界面

(2) 可以在 Server 下拉列表框中选择要添加的服务器的类型。添加 Tomcat 时，存在两个可选项：Tomcat 5.0 与 Tomcat 5.5。在此选择 Tomcat 5.5 选项。

(3) 在 Name 文本框中为要添加的容器设置名称，该值即为在运行环境窗口中显示的内容（如图 14-2 所示的“Bundled Tomcat (5.5.9)”）。在此将该值设置为“MyTomcat 5.5”。单击“Next”按钮，如图 14-16 所示。

(4) 在“Catalina”文本框中设置要添加的 Tomcat 的路径（Tomcat 的安装目录）。在“Username”与“Password”文本框中填写 manager 的用户名和密码。设置完成后，单击“Finish”按钮完成 Tomcat 的添加。此时运行环境窗口中内容如图 14-17 所示。

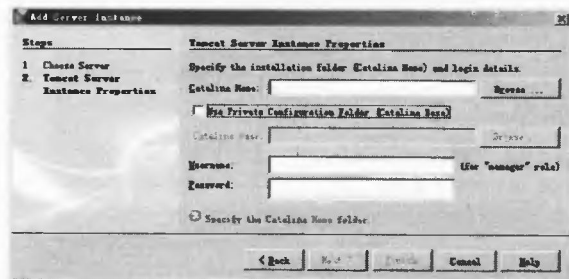


图 14-16 选择 Tomcat 目录

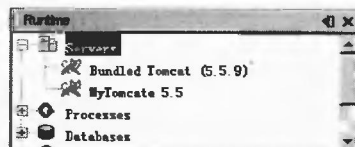


图 14-17 成功添加 Tomcat

此时已成功地向 NetBeans 中添加了 Tomcat。

14.3.2 配置 Tomcat

向 NetBeans 中成功添加 Tomcat 后,还可以对添加的 Tomcat 进行配置,按如下步骤进行。

(1) 在运行时环境窗口中右键单击“**MyTomcate 5.5**”节点,在弹出的菜单中选择“**Properties**”选项,打开如图 14-18 所示窗口。

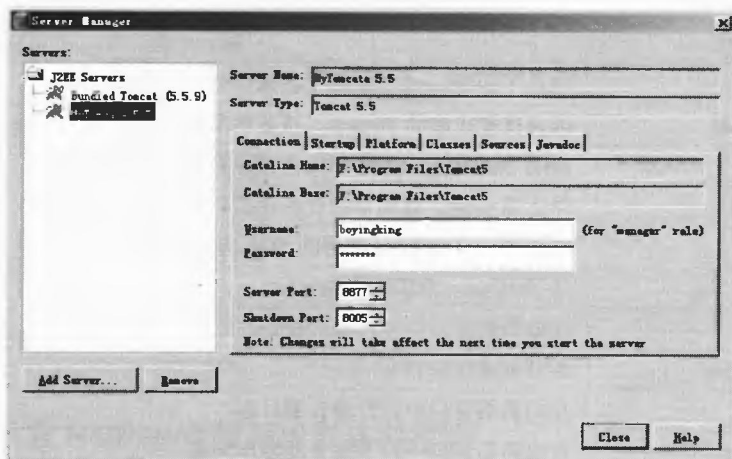


图 14-18 设置 Tomcat 属性对话框

(2) 可以在窗口中填写启动 NetBeans 所需要的用户名和密码以及 Tomcat 的启动端口与停止端口(以上内容必须与安装 Tomcat 时设定的内容一致)。

(3) 设置完成后,单击“**Startup**”标签,打开如图 14-19 所示标签页。在这里设置与启动有关的选项,还可以设置调试时共享内存区域的名称与端口号码。

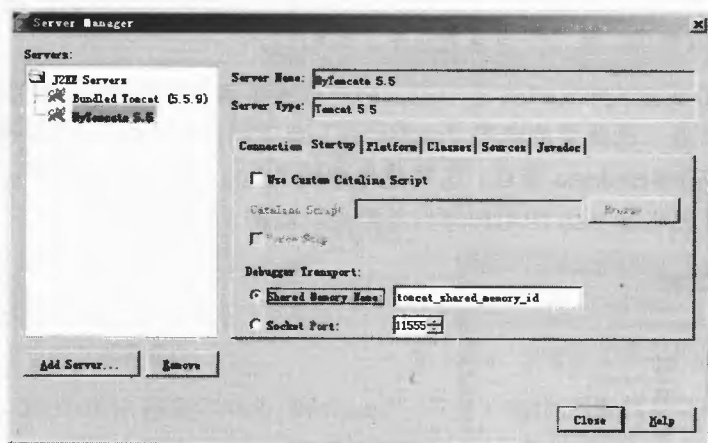


图 14-19 设置启动选项

(4) 单击“**Close**”按钮完成设置。

还可以在其他几个标签窗口中设置与 JDK 有关的一些内容,如设置 Tomcat 使用的
提示 JDK,向类路径中添加类文件或者 jar 文件等。这些设置都很简单,这里就不给出详细的说明了,读者可以自己试着进行这些操作。

14.3.3 操作 Tomcat

配置好 Tomcat 以后,就可以在 NetBeans 中操作 Tomcat。在运行时环境窗口中右键单击 MyTomcate 节点,通过选择右键菜单中的选项,可以完成不同的功能,表 14-1 给出了各个选项的说明。


表 14-1

右键菜单的选项及功能说明

选 项	说 明
Start	启动 Tomcat。该菜单项仅在未启动 Tomcat 时可用
Start in Debug Mode	以调试模式启动 Tomcat。该菜单项仅在未启动 Tomcat 时可用
Restart	启动 Tomcat。该菜单项仅在已经启动 Tomcat 时可用
Stop	停止 Tomcat。该菜单项仅在已经启动 Tomcat 时可用
Refresh	刷新 Tomcat 下已经部署的 Web 应用列表
Remove	从 NetBeans 中移除该 Tomcat
Edit Server.xml	在编辑器窗口中打开 Server.xml 文件
View Admin Console	打开管理员控制台
View Server Log	在编辑器窗口中打开服务器日志
View Server Output	在编辑器窗口中打开服务器输出记录
Properties	打开属性对话框

按照如下步骤完成对 Tomcat 的操作。

(1) 在运行时环境窗口中右键单击“MyTomcate 5.5”节点,在弹出的菜单中选择“Start”选项来启动 Tomcat。成功启动 Tomcat 后,项目窗口中内容如图 14-20 所示。

 **说明** Web Application 是部署的 Web 应用程序的根,相当于前面提到的 webapps 目录。其下的各个子节点,除“/”节点外,其他的每个节点都对应着已经部署到 Tomcat 中的 Web 应用程序。而“/”节点对应着“ROOT”。

(2) 右键单击 Web Application 子节点,选择右键菜单中的选项可以进行相应的操作。例如选中/balancer 节点,选择菜单中的“undeploy”选项则会撤销对 balancer 应用程序的部署。

(3) 右键单击/jsf-cardemo 节点,选择菜单中的“Stop”选项,则会在/jsf-cardemo 节点后面出现[stopped],表明该 Web 应用程序当前停止。如图 14-21 所示。

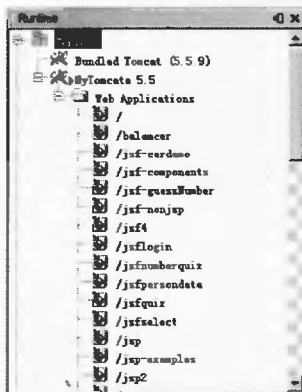


图 14-20 启动 Tomcat 后的项目窗口

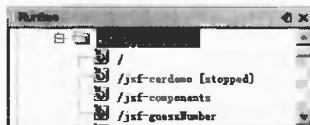


图 14-21 停止 jsf-cardemo

(4) 右键单击/节点, 选择菜单中的“Open in Browser”选项, 则会使用默认的浏览器打开该 Web 应用程序, 如图 14-22 所示。

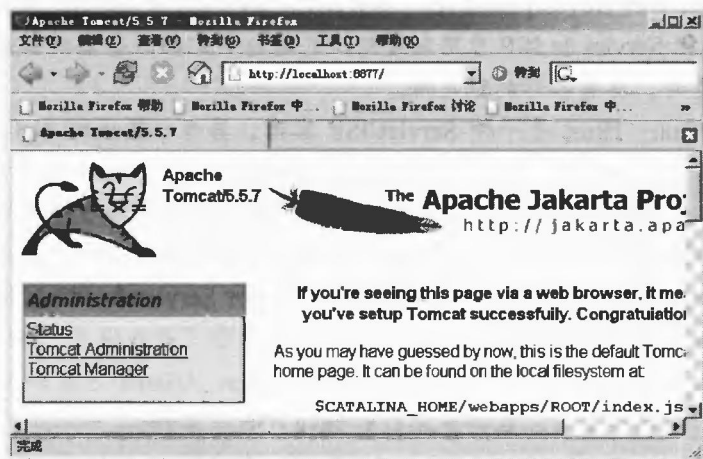


图 14-22 使用浏览器打开 Web 应用程序

14.3.4 设置 NetBeans 使用的默认浏览器

在 NetBeans 中运行 Web 程序时, 默认情况下, NetBeans 会在 IE 中打开程序的欢迎页面。如果想改变 NetBeans 使用的默认浏览器, 可以按照如下方式对 NetBeans 进行配置。

(1) 选择主菜单中“Tools”/“Options”选项, 在弹出的窗口中单击左下角的“Advanced Options”按钮, 出现高级自定义设置窗口, 如图 14-23 所示。

(2) 依次展开“IDE Configuration”/“Server and External Tool Settings”/“Web Browsers”节点, 如图 14-24 所示。

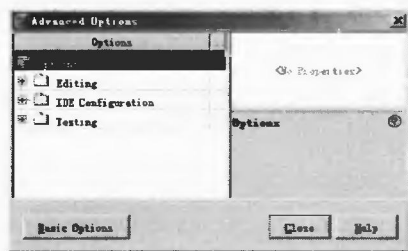


图 14-23 高级自定义设置窗口

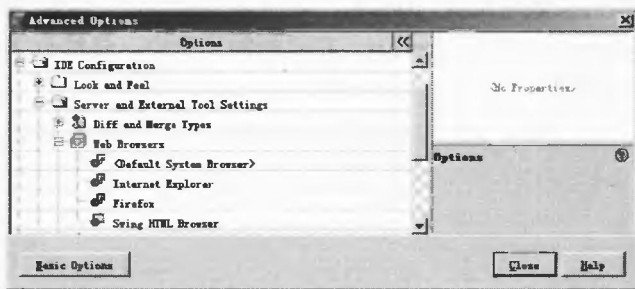


图 14-24 设置默认浏览器窗口

(3) 在图 14-24 中可以看到“Web Browsers”节点下列出了当前系统中可以使用的浏览器。选中要使用的浏览器, 单击“Close”按钮即可完成设置。

完成了上述步骤, 就已经成功的设置了 NetBeans 使用的默认浏览器。

14.4 其他 Web 服务器简介

虽然最终运行 Servlet 的往往是商业级的服务器, 但是开始学习的时候, 用一个能够在

PC 机上运行的免费系统进行开发和测试也足够了。除了 Tomcat 之外还有其他一些比较流行的 Web 服务器,下面列出了当前最受欢迎的几种。

- **JavaServer Web Development Kit (JSWDK):** JSWDK 是 Servlet 和 JSP 的官方参考实现,可以把 Servlet 和 JSP 应用部署到正式运行它们的服务器之前进行测试。JSWDK 也是免费的,具有很好的稳定性。
- **Allaire JRun:** JRun 是一种 Servlet/JSP 容器,其可以集成到 Netscape Enterprise、FastTrack Server、IIS、Microsoft Personal Web Server、版本较低的 Apache、O'eilly 的 WebSite 或 StarNine Web STAR。免费的 JRun 最多支持 5 个并发连接,而商业版本中不存在这个限制,同时商业版中增加了远程管理控制台之类的功能。
- **New Atlanta 的 ServletExec:** ServletExec 是一种 Servlet 和 JSP 容器,它可以集成到大多数流行的 Web 服务器。ServletExec 可以免费下载和使用,但许多高级功能和管理工具只有在购买了许可之后才可以使用。New Atlanta 还提供一个免费的 Servlet 调试器,该调试器可以在许多流行的 Java IDE 下工作。
- **Gefion 的 LiteWebServer (LWS):** LWS 是一个支持 Servlet 和 JSP 的免费小型 Web 服务器。Gefion 还有一个免费的 WAICoolRunner 插件,利用该插件可以为 Netscape FastTrack 和 Enterprise Server 增加对高版本的 Servlet 和 JSP 的支持。
- **Sun 的 Java Web Server:** 该服务器全部使用 Java 语言编写而成,是率先提供 Servlet 2.1 和 JSP 1.0 规范完整支持的 Web 服务器之一。虽然 Sun 现在已转向 Netscape/I-Planet Server,不再发展 Java Web Server,但该服务器仍旧是一个广受欢迎的 Servlet 与 JSP 的学习平台。

14.5 小结

Tomcat 是一个开源的 JSP/Servlet 容器,也是当前比较流行的 JSP/Servlet 容器。本章讲述了如何安装与配置 Tomcat,如何向 Tomcat 中部署 Web 应用程序,并详细地讲述了如何在 NetBeans 中使用 Tomcat。在后面的几章中,将结合 NetBeans 与 Tomcat 讲述如何进行 Web 项目的开发。

第 15 章

JSP 技术及应用

随着 Internet 技术应用的不断发展,尤其是电子商务等应用的出现,静态 HTML 页面已经不能满足人们的需求,因此出现了大量用来处理包含动态内容的 Web 页面技术。JSP 是处理包含动态内容的 Web 应用经常使用的一种技术,随着时间的推移将越来越流行。本章将对与 JSP 有关的内容进行讨论,主要包括如下内容:

- JSP 概述;
- 在 NetBeans 中开发第一个 JSP 程序;
- JSP 的模板元素;
- JSP 的脚本元素;
- JSP 的注释;
- JSP 的指令元素;
- JSP 的动作元素;
- 常用的 JSP 内建对象;
- 在 NetBeans 中开发完整的 JSP 程序。

15.1 JSP 概述

本节将介绍 JSP 技术,通过与 ASP 和 PHP 的比较了解 JSP 的优势。主要包括如下内容:

- JSP 技术介绍;
- ASP 与 JSP;
- JSP 的优势。

15.1.1 JSP 技术介绍

JSP (Java Server Pages) 是一种动态网页技术标准。在传统的 HTML 文件 (*.htm,*.html) 中加入符合一定规范的 Java 程序片段 (Scriptlet) 和 JSP 标记 (tag) 就构成了 JSP 网页 (*.jsp)。

Web 服务器在遇到访问 JSP 网页的请求时,首先执行其中相应的 Java 程序片段,然后将

执行结果以 HTML 格式返回给客户。程序片段可以是数据库操作、重新定向网页以及发送 email 等建立动态网页所需要的功能。

所有程序片段操作都在服务器端执行,通过网络传送给客户端的仅仅是程序片段运行得到的结果。这使得程序对客户浏览器的要求降到最低,可以实现无 Plugin、无 ActiveX、无 Java Applet 甚至无 Frame。以下是使用 JSP 的简单例子的代码。

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=GBK">
    <title>JSP Example</title>
  </head>
  <body>
    <h1>你好, JSP! </h1>
    <h1>这是一个使用 JSP 的例子! </h1>
  </body>
</html>
```

需要指出的是,上述代码并没有使用任何的 Java 程序段或者 JSP 标签,而仅仅使用了一些 HTML 标记。如果将上述代码存为.html 格式的文件,则可以直接使用浏览器打开该文件。将上述文件保存为 FirstJsp.html,使用浏览器打开该文件,则浏览器中内容如图 15-1 所示。

如果将上面给出的代码保存为.jsp 格式的文件,则不可以直接双击该文件来查看实际的运行效果,而需要将其部署到 JSP 容器中(如 Tomcat)才能查看。



注意

本书主要讲述如何在 NetBeans 中开发 JSP 和 Servlet 应用,并通过 NetBeans 将这些应用部署到 Tomcat 中,不会过多讲述如何单独使用 Tomcat 来部署 Web 应用程序。

当第一次请求 JSP 页面时,容器首先会将该 JSP 文件编译为 Servlet 类文件(该类实现了 javax.servlet.jsp.JspPage 接口,以后请求时就不会再编译该文件),然后容器会加载并运行这个 Servlet 类,最后将产生的结果发送到客户端。这一过程的示意图如图 15-2 所示。

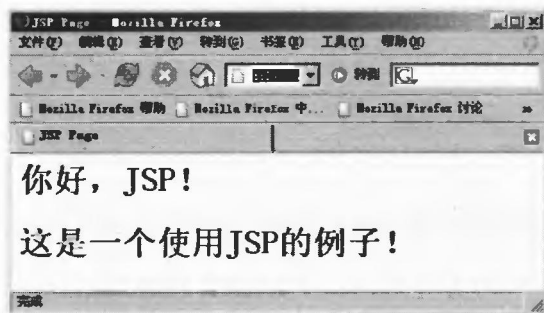


图 15-1 使用浏览器打开 HTML 文件

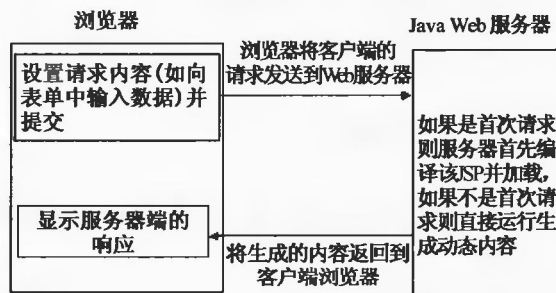


图 15-2 客户端请求与响应的过程

能够处理包含动态内容的 Web 应用技术有很多种,下面介绍另外两种常用的技术。

15.1.2 ASP 与 PHP

ASP 是 Microsoft 公司开发的一种处理动态页面 Web 应用的技术。更确切的说,ASP 是一个中间件,它将 Web 上的请求转入到一个解释器中,在该解释器中对所有的 ASP 的 Script

进行解释执行。

但是使用 ASP 时, Windows 本身的所有问题都会累加到它身上。ASP 通过使用 COM 组件使其功能变得强大, 但这样的强大由于 Windows NT 系统最初的设计问题而会引发更多的安全问题。图 15-3 给出了 ASP 的请求与响应过程。

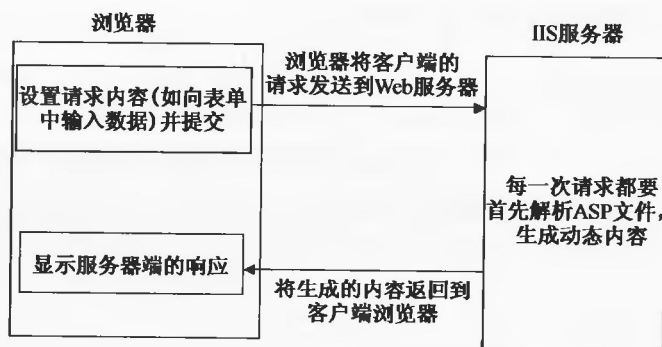


图 15-3 ASP 的请求与响应过程

提示 比较 ASP 与 JSP 的请求与响应过程图, 可以看到对于 ASP, IIS 服务器对于每一次请求都要解析 ASP 文件, 而 JSP 仅在第一次请求该文件时进行该项工作, 大大提高了速度。

PHP 是一种 HTML 内嵌式的处理包含动态页面的 Web 应用技术, 它混合了 C、Java、Perl 等语言的语法。PHP 与 HTML 有很好的兼容性, 并且提供了标准的数据库接口, 具有很好的跨平台性。但是 PHP 环境的配置非常复杂。PHP 的每一项扩充都会用到许多的外部应用库, 缺少正规的企业级支持与商业支持并且难于实现商业化的开发。所有开发都是基于脚本技术完成, 所有的源代码都无法编译。

15.1.3 JSP 的优势

与上面列出的这两种技术相比, JSP 主要具有如下优点。

- 运行速度快: 仅在第一次请求时进行编译、加载。
- 一次编写、到处运行。在不同平台上使用 JSP 时, 代码不用做任何修改。
- 系统的多平台支持: 基本上可以在所有平台上的任意环境中开发 JSP, 在任意的环境中进行部署, 在任意的环境中进行扩展。
- 强大的可伸缩性: 可以仅仅包含在一个 War 文件中, 完成比较简单的功能; 也可以在容器的支持下用在企业级应用中, 完成比较复杂的功能。
- 具有 Java 的所有强大的功能。
- 多样化开发工具的支持: 可以使用很多不同的开发工具进行开发, 并且很多开发工具都是免费的, 其中的很多工具可以运行于不同的平台之上(NetBeans 就是这样的开发工具)。

15.2 在 NetBeans 中使用 JSP

NetBeans 为开发与使用 JSP 提供了很大的便利。在 NetBeans 中可以很方便地创建、部署以

及运行 JSP 程序。本节将为读者介绍如何在 NetBeans 创建与运行 JSP 程序，主要包括如下内容：

- 在 NetBeans 中创建 Web 项目；
- 在 NetBeans 中配置 Web 项目；
- 在 NetBeans 中开发 JSP 程序。

15.2.1 在 NetBeans 中创建 Web 项目

在 NetBeans 中 Web 应用是以 Web 项目的形式存在的，本小节将为读者介绍如何在 NetBeans 中创建 Web 项目。启动 NetBeans，按照如下步骤完成该例。

(1) 依次选择主菜单中的“File”/“New Project”选项，出现如图 15-4 所示窗口。

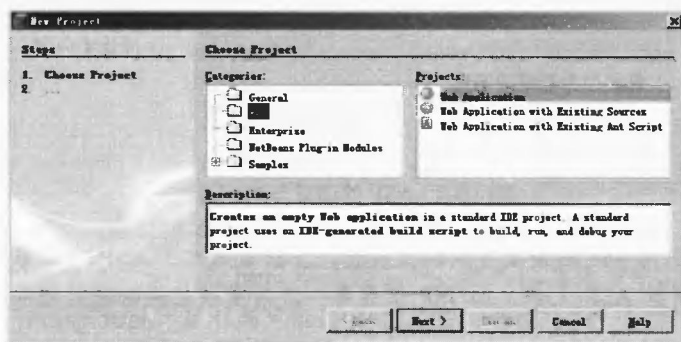


图 15-4 新建项目窗口

(2) 在“Categories”列表框中选中“Web”选项，并在“Projects”列表框中选中“Web Application”选项，单击“Next”按钮，如图 15-5 所示。

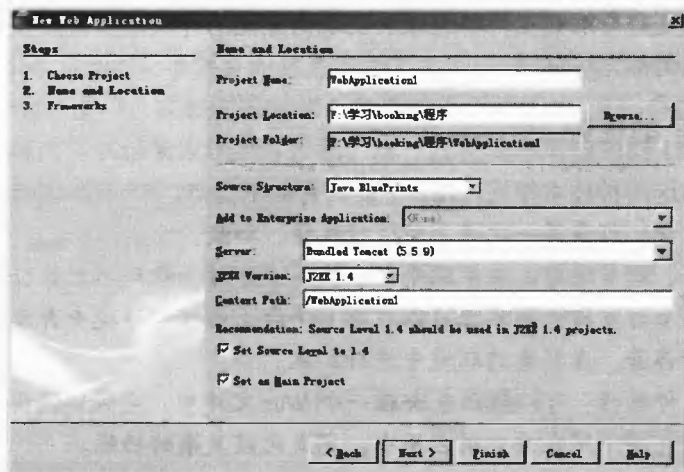



图 15-5 设置项目窗口

(3) 将“Project Name”的值修改为 DemoJspApplication，该值即为新创建的项目的名称。“Source Structure”下拉列表框中给出了可以用来创建 Web 应用的资源，有两个可选值：Java BluePrints 与 Jakarta，在这里选定 Jakarta。

 **说明** “Project Location”与“Project Folder”用来指定项目的位置和存放该项目使用的文件夹。

(4) “Server”下拉列表框中列出了已经安装的 Web 服务器，可以从中选择要使用的服务器，在这里使用前面添加的 MyTomcate 5.5。将“Context Path”的值修改为 DemoJspApplication，该值即为访问 Web 应用的上下文路径，其余选项均使用系统默认值。

(5) 单击“Next”按钮继续设置，打开如图 15-6 所示窗口。在这里可以设置要使用的 Web 框架。

(6) 选择完成后，单击“Finish”按钮完成项目的创建。此时项目窗口中内容如图 15-7 所示。

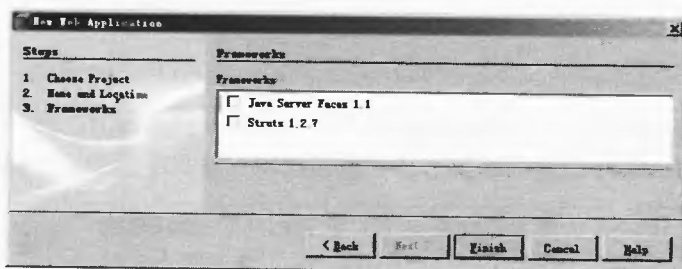


图 15-6 设置使用的框架

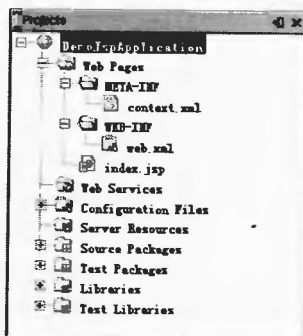


图 15-7 项目窗口中内容

此时就已经成功地在 NetBeans 中创建了一个 Web 项目。

15.2.2 在 NetBeans 中配置 Web 项目

在本节中，将主要讲述如何在 NetBeans 中配置 Web 项目。从图 15-7 中可以看到，“Web Pages”节点下包含 3 个子节点：“META-INF”、“WEB-INF”和“index.jsp”。

index.jsp 节点对应着新建项目时自动添加的 jsp 文件，该文件是此 Web 项目的默认欢迎文件。META-INF 下的 context.xml 是该 Web 应用的上下文配置文件。双击 context.xml 节点打开该文件，在编辑器窗口中会出现如下代码。

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/DemoJspApplication"/>
```

WEB-INF 下的 web.xml 是该 Web 应用的配置文件，双击 web.xml 打开该文件。编辑器窗口中内容如图 15-8 所示。

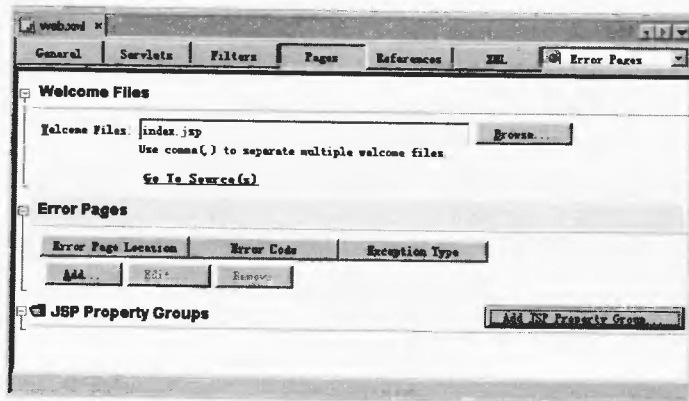


图 15-8 设置 Pages 选项

单击“General”标签可以打开如图 15-9 所示标签页。在这里可以设置该 Web 应用的显示名称、描述内容以及超时时间，还可以设置上下文参数以及 Web 应用的监听器等内容。

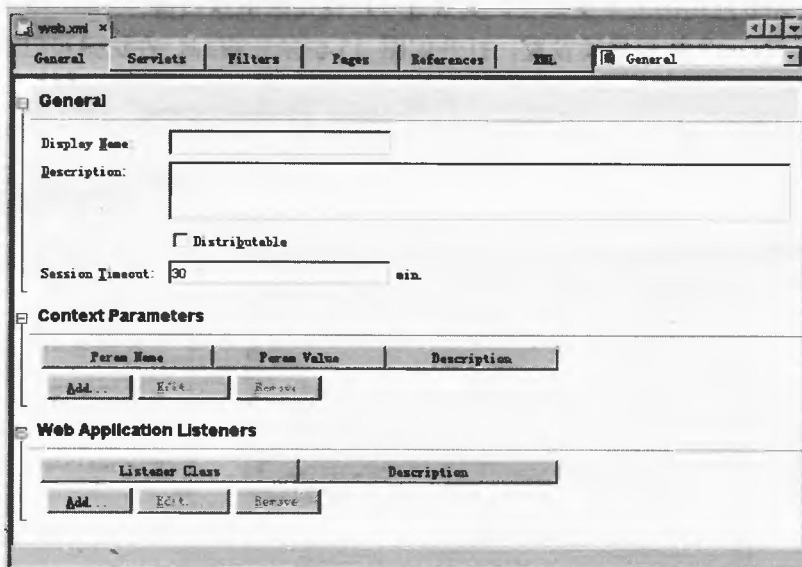


图 15-9 General 窗口

单击“References”标签可以打开如图 15-10 所示窗口。在该页面中可以设置与资源有关的一些内容，包括资源引用、资源环境引用、EJB 引用和消息目标引用等。

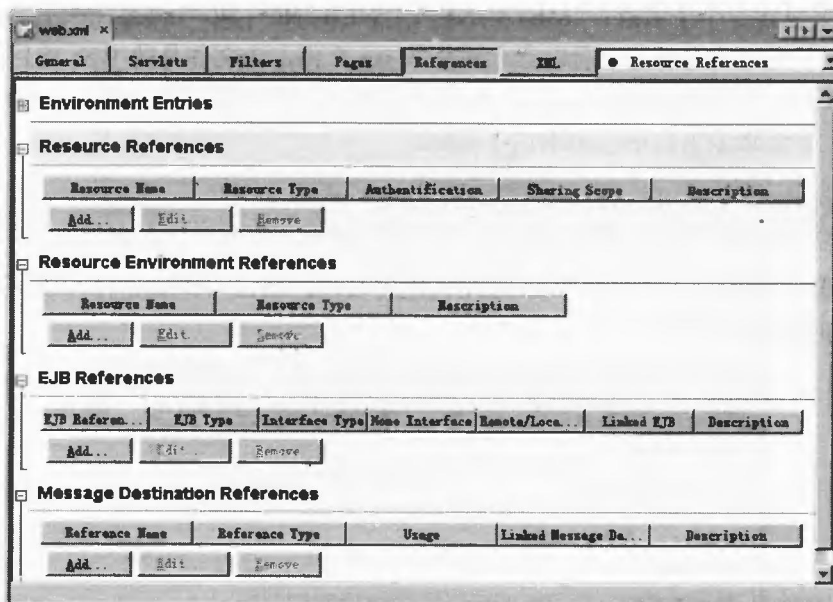


图 15-10 References 窗口

单击“XML”按钮，编辑器中会出现如下代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns=http://java.sun.com/xml/ns/j2ee
```



```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

<session-config>
  <session-timeout>
    30
  </session-timeout>
</welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</session-config>
</web-app>
```

以上代码即为 web.xml 文件的内容。在前面几个窗口中所设置的任何内容，最终都会在该编辑器中显示出来。当然，也可以在编辑器窗口中手工修改该文件的内容来进行配置。

提示 在 Servlet 与 Filters 窗口中设置的内容都与 Servlet 有关，在 Servlet 一章中会为读者进行详细的说明。

完成上述步骤，就已经成功地在 NetBeans 中创建了一个 Web 应用，并对其进行了配置。

15.2.3 在 NetBeans 中开发 JSP 程序

前两小节介绍了如何在 NetBeans 中创建与配置 Web 项目。本节将通过一个例子介绍如何在 NetBeans 中开发 JSP 程序。打开前面创建的项目“DemoJspApplication”，按照如下步骤完成该例。

(1) 在项目窗口中右键单击“Web Pages”节点，依次选择菜单中的“New”/“JSP”选项，出现如图 15-11 所示窗口。

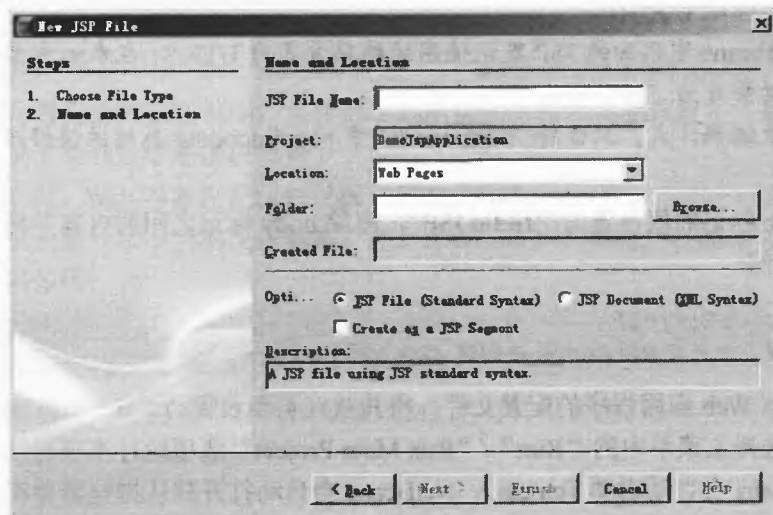


图 15-11 添加 JSP 窗口

(2) 在该窗口中，可以设置与要添加的 JSP 有关的一些选项，表 15-1 列出了各个选项的

说明以及在此设置的值。

表 15-1 选项说明以及设置值

选 项	说 明	设 置 值
JSP File Name	要添加的 JSP 文件名称	HelloJsp
Location	该 JSP 文件的存放位置	默认值
Folder	存放该 JSP 文件夹的名称	默认值
JSP File (Standard Syntax)	标准的 JSP 文件	默认值
JSP Document (XML Syntax)	XML 文件	默认值

(3) 按照表 15-1 的设置, 选定相应内容。设置完成后单击“Finish”按钮结束设置。此时在项目窗口中“Web Pages”节点下会出现一个名称为“HelloJsp.jsp”的节点。双击该节点, 编辑器中非注释部分代码如下:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; >
    <title>JSP Page</title>
</head>
<body>
    <h1>JSP Page</h1>
</body>
</html>
```

对这段代码作如下说明:

- 在 Netbeans 中添加的 JSP 默认使用的编码方式为 UTF-8, 在本例中需要将其编码方式手动修改为“gb2312”。
- 要修改编码方式, 只需将<%@page %>中 pageEncoding 属性的值修改为“gb2312”即可。

(4) 将 title 标记的值设置为“Hello JSP”, 删除 body 标记之间的内容并将下列代码添加到 body 标记之间:

```
<h1>Hello JSP!</h1>
<h1>你掌握了如何在 NetBeans 中创建 JSP 了吗? </h1>
```

(5) 修改该 Web 应用程序的配置文件, 将其欢迎页面设置为“HelloJsp.jsp”。

(6) 依次选择主菜单中的“Run”/“Run Main Project”选项运行本项目(如果没有启动 Tomcat, NetBeans 会自动启动 Tomcat), NetBeans 会自动打开默认浏览器并在浏览器中请求该 JSP, 浏览器中内容如图 15-12 所示。

在本例中使用了一些 JSP 标记, 如<%@page %>。在本节后面的部分, 将会对 JSP 的常用标记进行详细介绍。

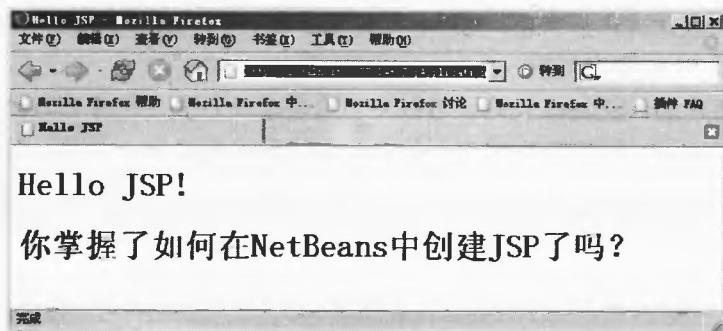


图 15-12 使用默认浏览器浏览 JSP

15.3 JSP 的模板元素

模板元素是指 JSP 中包含的静态 HTML 或 XML 的内容，这些元素对 JSP 的显示是非常重要的。但是 JSP 开发人员可能并不会特别关心这部分内容，该项内容主要由网页的美工人员来完成。下列代码使用了 JSP 模板元素：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP </title>
</head>
<body>
    <h1>JSP World</h1><br>
    <h2> JSP 模板元素应用示例</h2>
</body>
</html>
```

这些模板元素说是网页的框架，直接影响到网页的结构和美观程度。一般情况下，会使用可视化的开发工具开发这些模板元素。

在编译 JSP 时，Web 容器将把这些模板元素编译到 Servlet 中。当客户请求此 JSP 时，容器会将这些模板元素一字不差的发送到客户端。比如<html>、<head>在编译成 Servlet 时，会使用如下的代码替代：

```
out.write("<html>\r\n\t");
out.write("<head>\r\n\t");
```

本书不对模板元素所包含的 HTML 或者 XML 进行详细讲述，需要用到的读者可以查阅相关的参考资料。

15.4 JSP 的脚本元素

JSP 的脚本元素是 JSP 代码中使用最频繁的元素，通常由 Java 语言编写代码，允许声明

变量和方法以及对表达式进行求值。JSP 中主要包括如下 3 种脚本元素：声明（Declaration）、表达式（Expression）与程序片段（Scriptlets）。

JSP 的声明部分是一段 Java 代码，可以声明属性或者方法。声明后的属性或方法可以在 JSP 中的任意地方使用，相当于 Java 类中的成员方法和成员变量。声明的格式如下：

```
<%! Variable declaration %>
```

或者

```
<%! method declaration(paramType param.....)%>
```

以下代码声明了几个变量。

```
<%!  
    String name=request.getParameter("name");  
    String password=request.getParameter("password");  
    String age=request.getParameter("age");  
    String gender=request.getParameter("gender");  
%>
```

JSP 表达式的值在 JSP 的请求处理阶段被计算，所得结果被转换成为字符串并与模板元素相结合。客户端请求 JSP 时，表达式的计算结果所处的位置与表达式在 JSP 中的位置一致。

表达式的语法如下：

```
<%= java expression %>
```

表达式本身必须有一个返回值或者其本身就是一个可以使用 toString 方法的对象。实际上，编译 JSP 文件时，表达式的值被转换为 out.println() 方法中的内容。比如：

```
<%= "你好！JSP"%>
```

相当于 JSP 页面中的

```
out.println("你好！JSP");
```

在适当的地方使用表达式，可以使程序变得更加简洁。以下代码使用表达式输出当前时间：

```
<%@ page contentType="text/html; charset=gb2312" %>  
<%=new java.util.Date().toLocaleString() %>
```

JSP 程序片段是 JSP 中使用最为频繁的脚本元素，其实质上是一段可以在处理请求期间内执行的 Java 代码。JSP 程序片段包括在<%、%>之间。其可以产生输出，并将输出发送到客户端也可以是一些流程控制语句，当然还可以包含一些合法的 Java 注释。

JSP 程序片段的语法如下：

```
<% java code %>
```

以下代码使用 JSP 程序片段在客户端输出一些内容：

```
<%!  
    int maxNum=10;
```

```
%>
<h4>以下将输出从 0 到 maxNum 之间的内容</h4>
<%
    for(int i=0;i<maxNum;i++){
        out.println("<br>" + i);
    }
%>
```

可以看到，程序片段中的代码与普通的 Java 应用程序中的代码没有什么不同。

提示 JSP 程序片段相当于 JSP 程序的主方法，用来对客户请求进行服务。当然，在服务的过程中可以调用声明部分声明的方法和变量。

15.5 JSP 注释

在 JSP 中可以使用多种注释。常用的有显式注释、隐式注释以及程序片段中的注释。显式注释主要是指 HTML/XML 注释。当请求包含显式注释的 JSP 页面时，可以通过查看客户端 HTML 的源代码看到该注释。显式注释的语法如下：

```
<!-- 注释内容-->
```

下面给出了在 JSP 中使用显式注释的完整代码（comment1.jsp）：

```
<%@page contentType="text/html"%>
<%@page pageEncoding="GBK"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=GBK">
        <title>JSP </title>
    </head>
    <body>
        <h2>这是一个使用 HTML 注释的界面</h2>
        <h2>可以通过查看源文件来查看注释</h2>
        <!-- 这里是注释内容，你看到了吗？ -->
    </body>
</html>
```

将 comment1.jsp 保存到以前安装的 Tomcat 的 %TOMCAT_HOME%\webapps\ROOT 目录中，启动 Tomcat，在浏览器的地址栏中输入如下内容：

```
http://localhost:8877/comment1.jsp
```

8877 为上一章中安装的 Tomcat 的端口号。读者也可以将该文件保存到系统中已经安装的其他 Tomcat 的 webapps\ROOT 目录中，在浏览器中进行访问时，只需修改端口号即可。

此时，浏览器中内容如图 15-13 所示。这里也可以查看该页面的源代码，如图 15-14 所示。

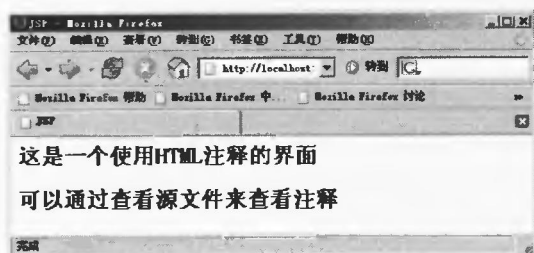


图 15-13 请求包含显式注释的 JSP 页面

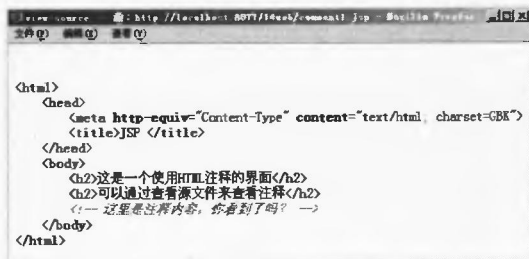


图 15-14 查看包含显式注释页面的源代码

从图中可以看到, 在 JSP 文件中使用了注释——“`<!--这里是注释内容, 你看到了吗? -->`”, 可以通过查看 HTML 页面的源代码来查看该注释。该注释内容与 JSP 文件中的注释内容一致。另外, 还可以在显式注释中使用表达式。例如在 JSP 文件中添加如下注释:

```
<!-- 当前时间为: <%= (new java.util.Date()).toLocaleString() %> -->
```

在客户端的 HTML 源代码显示为:

```
<!-- 当前时间为: 2006-4-15 16:40:00 -->
```

以上所示注释与前面所给出的注释很相似, 惟一不同之处就是在浏览器中查看该页面的源代码时, 内容不固定。该内容根据 JSP 运行情况的不同而有所不同。

还可以在 JSP 页面中使用隐式注释。用隐式注释标记的字符在编译 JSP 时会被忽略。隐式注释的语法如下:

```
<%-- comment--%>
```

下面给出了使用隐式注释的完整代码 (comment2.jsp):

```
<%@page contentType="text/html"%>
<%@page pageEncoding="GBK"%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=GBK">
    <title>隐式注释</title>
  </head>
  <body>
    <h2>这是一个使用隐式注释的界面</h2>
    <h2>不能通过查看源文件来查看隐式注释</h2>
    <%-- 这里是使用隐式注释的例子 --%>
    当前时间为<%= (new java.util.Date()).toLocaleString() %>
  </body>
</html>
```

将 comment2.jsp 保存到 %TOMCAT_HOME%\webapps\ROOT 目录中, 启动 Tomcat (如果此时 Tomcat 没有启动), 然后在浏览器的地址栏中输入如下内容:

```
http://localhost:8877/comment2.jsp
```

此时浏览器中内容如图 15-15 所示。查看该页面的源代码, 效果如图 15-16 所示。

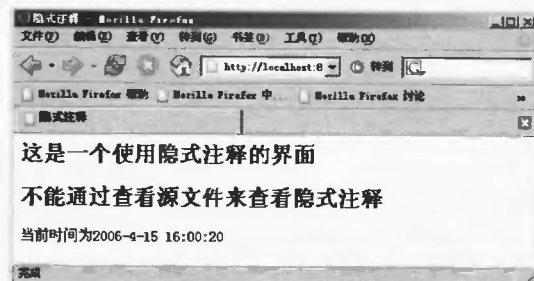


图 15-15 请求包含隐式注释的 JSP 页面

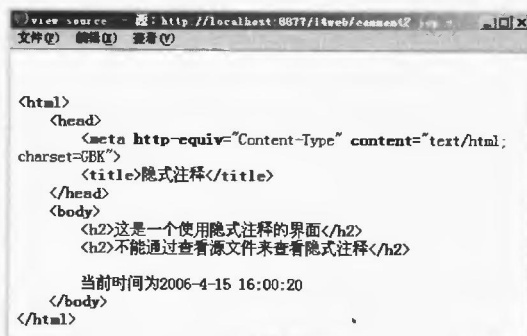


图 15-16 查看包含隐式注释页面的源代码

从图中可以看到，它使用了隐式注释“<!--这里是使用隐式注释的例子-->”，从而通过查看客户端 HTML 的源代码并不能看到该注释。

除了隐式注释与显式注释之外，JSP 中还可以包括程序片段注释。由于程序片段中包含的是纯 Java 代码，所以在 Java 中的注释规则在程序片段中也能使用。

在 Java 中，使用“//”表示单行注释，使用“/**/”表示多行注释。例如使用“//”注释：

```
// count 是一个成员变量，其用来记录按钮被按下的次数  
private int count=0;
```

也可以使用“/**/”，代码如下：

```
/*  
 *count 是一个成员变量，  
 *其用来记录按钮被按下的次数  
 */  
private int count=0;
```

15.6 JSP 的指令元素

指令元素用于从 JSP 发送一条信息到 Web 容器中，为编译阶段提供了一些全局信息。指令元素并不向客户端产生任何输出，并且所有的指令都在 JSP 的整个文档范围内有效。

JSP 包括页面指令（page）、include 指令和 taglib 指令 3 种指令元素。本节将对这 3 种指令元素进行详细的介绍。

15.6.1 页面指令

页面指令用来定义影响到整个页面的许多重要属性。一个 JSP 页面中可以包含多个页面指令。除了 import 属性之外，其他的属性/值在同一页面中只能出现一次。页面指令的格式如下：

```
<%@ page attribute="value".....%>
```

其中，attribute 可以为 language、import、contentType、session、buffer、autoFlush、isThreadSafe、errorPage、isErrorPage 中的一个。

页面指令的详细语法如下：


```

<%@ [ language="java" ] ·
    [ import="package.class |package.*,...." ]
    [ contentType="TYPE,charset=CHARSET" ]
    [ session="True | false" ]
    [ buffer="none | 8kb | sizekb" ]
    [ autoFlush="True | false" ]
    [ isThreadSafe="True | false" ]
    [ info="text" ]
    [ errorPage="relativeURL" ]
    [ isErrorPage="True | false" ]
    [ extends="package.class" ]
    [ isELIgnored="True | false" ]
    [ pageEncoding="peiinfo" ]
%>

```

表 15-2 给出了这些属性的详细说明。

表 15-2 JSP 的页面属性及其说明

属 性	说 明	默 认 值	例 子
language	定义要使用的语言,目前只能使用“java”	java	language="java"
import	和普通的 Java 语言中的 import 意义一样,用来引入 JSP 中要使用的包或类。只是使用“,”隔开包或者类列表	默认忽略(即不引入其他的类或者包)	import="java.util.*java.swing.*java.awt.*"
session	指定 HTTP 会话中该页面是否参与	true	session="true"
buffer	指定到客户输出流的缓冲模式,如果是 none,则不缓冲,如果是指定数值,则输出用不小于该值的值进行缓冲	不小于 8kb,可以根据不同的服务器进行设置	buffer="64kb"
autoFlush	值为 true 时,到客户端的输出被刷新。如果值为 false,则缓冲区满时,会出现异常情况	true	autoFlush="true"
info	关于 JSP 页面的信息,定义一个字符串,可以使用 servlet 对象的 getServletInfo 获得	默认忽略	info="一个测试界面"
isErrorPage	表明当前页面是否为其他页的 errorPage 目标。如果设置为 true,则可以使用 exception 内建对象。如果设置为 false,则不可以使用 exception 内建对象	false	isErrorPage="false"
errorPage	定义此页面出现异常时调用的页面	默认忽略	errorPage="error/error.jsp"
isThreadSafe	设置 JSP 文件是否能够被多线程并发使用。如果设置为 true,则一个 JSP 可以同时处理多个用户请求;相反,如果设置为 false,则一次只能处理一个用户请求	true	isThreadSafe="true"
contentType	定义 JSP 字符编码和页面响应的 MIME 类型。TYPE=MIME TYPE; charset=CHARSET	TYPE=text/html charset=utf-8	contentType="text/html; charset=GBK"
pageEncoding	JSP 页面的字符编码	pageEncoding="ISO-8859-1"	pageEncoding="gb2312"
isELIgnored	是否忽略 EL 表达式。如果为 true,则忽略	默认值由 web.xml 描述文件的版本所确定,Servlet 2.3 以前的版本将会忽略	isELIgnored="true"

这些指令可以单独使用，也可以同时使用其中的几个。下面给出了使用页面指令的完整代码（UsePage.jsp）。

```
<%@ page import="java.util.*"
    session="true"
    buffer="12kb"
    autoFlush="true"
    info="测试 JSP 页面指令"
    isErrorPage="false"
    contentType="text/html; charset=gb2312"
%>
<%@ page isELIgnored="false"%>
<html>
  <head>
    <title>测试 JSP 的页面指令</title>
  </head>
  <body>
    <h2>这是一个测试 JSP 的页面指令的 web 应用</h2>
    当前时间<%=new Date().toLocaleString()%>
  </body>
</html>
```

将 UsePage.jsp 保存到 %TOMCAT_HOME%\webapps\ROOT 目录中，启动 Tomcat（如果此时 Tomcat 没有启动），然后在浏览器的地址栏中输入如下内容：

```
http://localhost:8877/UsePage.jsp
```

此时浏览器中内容如图 15-17 所示。

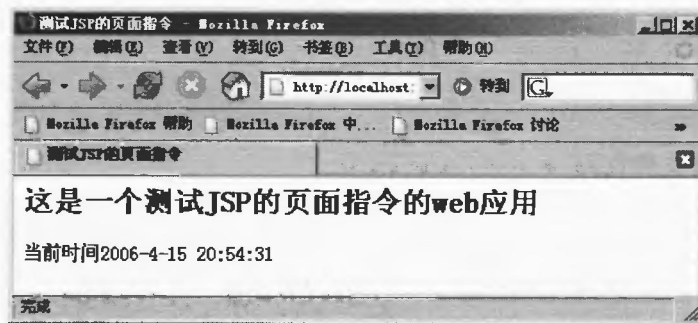


图 15-17 测试 JSP 页面指令

15.6.2 include 指令

include 指令用来通知容器当前 JSP 页面中内嵌的、在指定的位置上包含的资源文件。被包含的文件内容可以被 JSP 解析，这种解析发生在编译期间。使用 include 指令的语法如下：

```
<%@ include file="filename"%>
```

其中，filename 为要包含文件的文件名称，也可以包含路径。需要注意的是，一经编译，

其内容不可改变,这使其执行效率比较高。如果改变 filename 的内容,必须重新编译 JSP 文件,当然,这是自动完成的。

如果路径以 “/” 开头,那么路径主要参照 JSP 应用的上下文路径关系;如果路径是以文件名称或目录名称开头,那么该路径参照正在使用的 JSP 的当前路径。下面给出了使用 include 指令元素的完整代码 (UseInclude.jsp)。

```
<%@ page contentType="text/html;charset=gb2312" %>
<%@ include file="first.html" %>
<%@ include file="second.html"%>
<%@ include file="third.html"%>
```

first.html, second.html 与 third.html 是当前路径中存在的 HTML 文件,具体内容如下。

(1) first.html 文件代码

```
<html>
<head>
<title>include example</title>
</head>
<body>
<h2>this is the first page</h2>
</body>
</html>
```

(2) second.html 文件代码

```
<html>
<head>
<title>include example</title>
</head>
<body>
<h2>this is the second page</h2>
</body>
</html>
```

(3) third.html 文件代码

```
<html>
<head>
<title>include example</title>
</head>
<body>
<h2>this is the third page</h2>
</body>
</html>
```

将 UseInclude.jsp、first.html、second.html 以及 third.html 文件保存到 %TOMCAT_HOME%\webapps\ROOT 目录中,启动 Tomcat (如果此时 Tomcat 没有启动),然后在浏览器的地址栏中输入如下内容:

```
http://localhost:8877/UseInclude.jsp
```

此时浏览器中内容如图 15-18 所示。

`include` 指令将在编译 JSP 时插入一个包含文本或者代码的文件。当使用 `include` 指令时, 这个包含的过程就是静态的。

静态的包含就是指这个被包含的文件的内容将会被插入到包含它的 JSP 文件中。被包含的文件可以是 JSP 文件、HTML 文件、文本文件、`inc` 文件等。如果包含的文件中包含可执行代码, 那么这个包含的文件中的代码将会被执行。

如果仅仅使用 `include` 来包含一个静态文件, 那么被包含的文件的内容将会被插入到 JSP 文件中放 `<%@ include %>` 的地方。如在浏览器中查看图 14-26 所示页面的源代码, 内容如下:

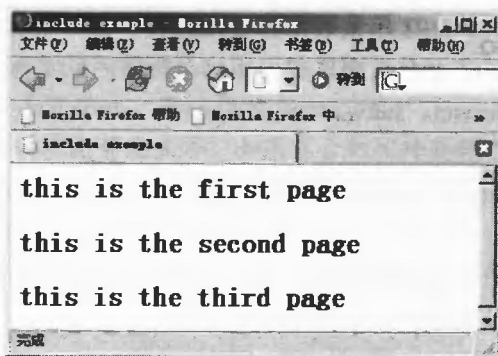


图 15-18 使用 JSP 的 `include` 指令

```
<html>
  <head>
    <title>include example</title>
  </head>
  <body>
    <h2>this is the first page</h2>
  </body>
</html>
<html>
  <head>
    <title>include example</title>
  </head>
  <body>
    <h2>this is the second page</h2>
  </body>
</html>
<html>
  <head>
    <title>include example</title>
  </head>
  <body>
    <h2>this is the third page</h2>
  </body>
</html>
```

使用 `include` 指令可以将一个复杂的 JSP 页面分成若干的简单部分, 这样就大大增加了 JSP 页面的管理性。当要对页面进行更改时, 只需要更改对应部分就可以了。

15.6.3 taglib 指令

`taglib` 指令允许页面开发者使用自定义标签, 在 JSP 页面里使用 `taglib` 指令的语法如下:

```
<%@ taglib uri="taglibURI" prefix="tagPrefix"%>
```

URI 用来表示标签描述符,即告诉容器如何找到标签描述文件和标签库。`tagPrefix` 定义了 JSP 页面里要引用这个标签时的前缀,这些前缀不可以是 `jsp`、`jspx`、`java`、`javax`、`sun`、`servlet`、`sunw`。

15.7 JSP 动作元素

与指令元素不同的,动作元素在请求处理阶段起作用。JSP 动作元素是使用 XML 元素语法写成的,其采用如下格式:

```
<prefix:tag attribute=value attribute-list.../>
```

或者采用如下格式:

```
<prefix:tag attribute=value attribute-list...>
.....
</ prefix:tag >
```

容器在处理 JSP 时,每当遇到这种动作元素,都根据其标记进行相应的处理。JSP 规范定义了一系列的动作标准,其使用 `jsp` 作为前缀。这些标准的动作不管容器是如何实现的,但是每个 Web 容器都必须提供这些操作类型。

标准动作作为页面设计者提供了基本的功能,其他的厂商可以自由提供替这些动作的非标准的动作以增强功能。标准的 JSP 动作元素有 `<jsp:useBean>`、`<jsp:setProperty>`、`<jsp:getProperty>`、`<jsp:param>` 等。

15.7.1 `<jsp:param>`与`<jsp:include>`动作元素

`<jsp:param>`操作以“名-值”对的形式为其他的标签提供附加信息,经常与`<jsp:include>`、`<jsp:forward>`和`<jsp:plugin>`一起使用。使用方式如下:

```
<jsp:param name="paramname" value="paramvalue"/>
```

其中, `name` 为属性相关的关键字, `value` 为属性的值。在后面的几个小节,将会结合其他的 JSP 动作指令讲述`<jsp:param>`的使用。

`<jsp:include>`操作允许在请求时间内,在现有的 JSP 页面内包含静态或动态资源。被包含的对象只有对 `JspWriter` 对象的访问权,并且其不能设置头或者 Cookie。如果页面输出是缓冲的,那么缓冲区的刷新要优先于包含页面的刷新。

此指令在运行上比`<%@ page include%>`指令的效率低,但是可以动态增加内容。

```
<jsp:include page="filename" flush="true"/>
//或者
<jsp:include page="filename" flush="true">
<jsp:param name="paramname" value="paramValue">
...
</jsp:include>
```

下面给出了`<jsp:include>`指令中各个属性的详细说明:

- `page="filename"`: 参数为一个相对路径或者代表相对路径的上下文表达式。如果路

径以“/”开头,则该路径是参照 JSP 应用的上下文关系路径。如果路径是以文件名或者目录名开头,则该路径是 JSP 当前正在使用的路径。

- `flush="True"`: 如果使用 `false`, 则不能自动刷新输出, 默认值为 `false`。
- `<jsp:include>` 标签允许在 JSP 中包含动态文件或者静态文件。如果包含的文件是静态的, 那么这种包含仅仅是将包含文件的内容加到 JSP 文件中去; 如果包含的文件是动态的, 那么该包含文件将会被 JSP 编译器执行, 执行的结果被加到 JSP 文件中去。
- 如果包含的文件是动态的, 则还可以使用 `<jsp:param>` 传递一个或者多个参数给被包含的页面。

提示 与 `<%@ include %>` 指令不同的是 `<jsp:include>` 包含的内容可以是动态改变的, 在执行时才确定, 而前者包含的内容一经编译就不能改变。

下面给出了使用 `include` 指令的程序的完整代码 (UseJspInclude.jsp):

```
<%@ page contentType="text/html; charset=GBK" %>
<html>
  <head>
    <title>使用 include 动作指令</title>
  </head>
  <body>
    以下是静态页面的内容
    <%@ include file="staticpage.html"%><br>
    以下是动态页面的内容
    <jsp:include page="synatic.jsp" flush="true">
      <jsp:param name="firstparam" value="<%=request.getParameter("name")%%" />
      <jsp:param name="secondparam"
        value="<%=request.getParameter("password")%%" />
    </jsp:include>
  </body>
</html>
```

`staticpage.html` 是一个静态 HTML 文件, 用来提交数据, 其代码如下:

```
<html>
  <head>
    <title>静态页面</title>
  </head>
  <body>
    <form action="UseJspInclude.jsp" name="myform" method="post">
      UserName:<input type="text" name="name"><br>
      Secret:<input type="text" name="password"><br>
      <input type="submit" value="OK" size="40">
    </form>
  </body>
</html>
```

`synatic.jsp` 用来动态输出 `UseJspInclude.jsp` 传递过来的参数, 其代码如下:

```
<%@ page contentType="text/html; charset=gb2312" %>
<h4>这是第一个参数<%=request.getParameter("firstparam") %></h2>
<h4>这是第一个参数<%=request.getParameter("secondparam") %></h2>
<h4>你掌握了 include 动作指令的使用方法了吗? </h3>
```

将 UseJspInclude.jsp、staticpage.html 及 synatic.jsp 保存到%TOMCAT_HOME%\webapps\ROOT 目录中, 启动 Tomcat, 然后在浏览器的地址栏中输入如下内容:

http://localhost:8877/UseJspInclude.jsp

此时浏览器中内容如图 15-19 所示。

从图中可以看到, synatic.jsp 页面的内容是动态变化的, 其内容由 firstparam 与 secondparam 决定。而 staticpage.html 的内容是不变的。

15.7.2 <jsp:forward>动作元素

<jsp:forward>操作用来将请求转发到另一个 JSP、Servlet 或静态资源文件。请求被转向的资源必须位于同 JSP 发送请求相同的上下文环境中。每当遇到此操作时, 就停止执行当前的 JSP, 转而执行被转发的资源。按以下的方式使用此操作。

```
<jsp:forward page="uri"/>
```

或者使用如下的方式使用 jsp:forward。

```
<jsp:forward page="uri">
  <jsp:param name="paramname" value="paramvalue">
  .....
</jsp:forward>
```

例如:

```
<jsp:forward page="/welcome.jsp"/>
```

还可以在 jsp:forward 中使用<jsp:param>, 向转发到的位置传递参数。

```
<jsp:forward page="/welcome.jsp">
  <jsp:param name="userID" value="hello"/>
  <jsp:param name="password" value="secret">
</jsp:forward>
```

下面给出了<jsp:forward>指令中各个属性的详细说明。

- page="uri": uri 可以为一个表达式或者一个字符串, 它用来指定要定向的文件或 URL。该文件可以是 JSP 文件、程序段或其他的能够处理 request 对象的文件。
- <jsp:param name="paramname" value="paramvalue">: name 属性用来指定参数名称, value 指定参数值。参数被发送到一个文件, 可以是一个值或者多个值, 但是该文件



图 15-19 使用 include 动作指令的 Web 应用

必须是动态文件。

<jsp:forward>的典型应用就是用户登录。成功验证用户权限以后,则跳转到相应的页面。下面给出了一个使用<jsp:forward>的程序的完整代码(UseJspForward.jsp)。如果验证成功,则跳转到 welcome.jsp 页面;如果不成功,则跳转到 login.jsp 页面进行重新验证。

```
<%@ page contentType="text/html; charset=gb2312" %>
<html>
  <head>
    <title>使用 forward 动作指令</title>
  </head>
  <body>
    <%
      String name=request.getParameter("name");
      String password=request.getParameter("password");
      if((name!=null&&(name.equals("terrywang"))
        &&(password!=null&&(password.equals("123456")))){
    %>
    <jsp:forward page="welcome.jsp">
      <jsp:param name="username" value="<%=name%>" />
    </jsp:forward>
    <%
    }else{
      %>
    <jsp:forward page="login.jsp">
      <jsp:param name="username" value="<%=name%>" />
    </jsp:forward>
    <%
    }
    %>
  </body>
</html>
```

login.jsp 文件中代码如下:

```
<%@ page contentType="text/html; charset=gb2312" %>
<html>
  <head>
    <title>用户登录</title>
  </head>
  <body>
    <form action="UseJspForward.jsp" name="myform" method="post">
      UserName:<input type="text" name="name"><br>
      Secret:<input type="text" name="password"><br>
      <input type="submit" value=" 确定 " >
    </form>
  </body>
</html>
```

welcome.jsp 文件中代码如下:

```
<%@ page contentType="text/html; charset=gb2312" %>
<h4>登录成功:</h4>
<h4>欢迎你, <%=request.getParameter("username") %></h4>
```

将 UseJspForward.jsp、login.jsp 及 welcome.jsp 文件保存到 %TOMCAT_HOME%\webapps\ROOT 目录中, 启动 Tomcat (如果此时 Tomcat 没有启动), 然后在浏览器的地址栏中输入如下内容:

http://localhost:8877/login.jsp

此时浏览器中内容如图 15-20 所示。如果在“户名”与“密码”文本框中输入内容不是“terrywang”或“123456”, 则仍会跳转到该页面重新进行验证。

在用户名与密码文本框中分别输入“terrywang”与“123456”, 单击“确定”按钮, 此时浏览器中内容如图 15-21 所示。

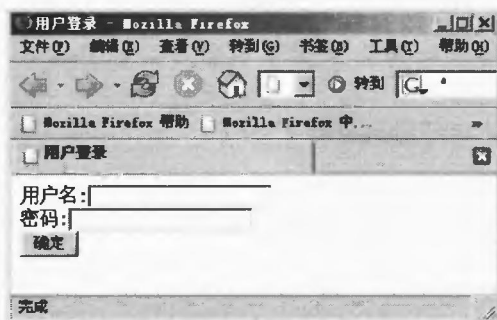


图 15-20 请求 login.jsp 页面

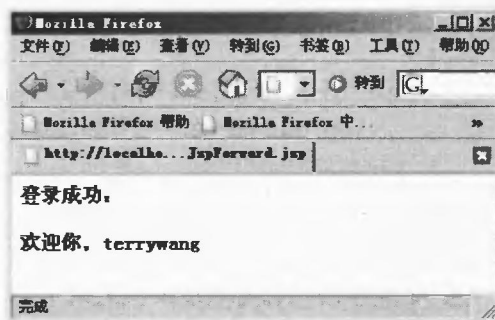


图 15-21 登录成功页面

本例的验证过程比较简单, 实际应用中一般都会通过连接数据库来进行验证, 几乎所有的登录验证都是这样的: 首先获得用户登录信息, 然后根据该信息进行验证, 如果验证成功, 则把视图跳转到登录成功后的页面, 反之把视图跳转到登录页面。

15.8 常用的 JSP 内建对象

JSP 为简化页面的开发提供了一系列内建对象。这些内建对象不需要由 JSP 开发人员创建, 而由容器创建和管理。在所有的 JSP 页面中都可以使用内建对象, JSP 有以下内建对象:

- out 对象;
- request 对象;
- response 对象;
- session 对象;
- pageContext 对象;
- application 对象;
- config 对象;
- page 对象;
- exception 对象。

在接下来的几节中，将会对其中常用的几个内建对象进行详细的介绍。

15.8.1 输出内建对象——out

out 内建对象被封装成 `javax.servlet.jsp.JspWriter` 类。它表示为客户打开的输出流，主要用来向客户输出数据。out 内建对象可以使用一些方法进行相应的操作，表 15-3 列出了一些常用的方法及说明。

表 15-3 out 内建对象的常用方法及说明

方 法	说 明
<code>print(String s)</code>	向客户端输出数据，该方法有多种重载方法
<code>println(String s)</code>	向客户端输出数据，并在后面添加一个空行。该方法有多种重载方法
<code>newLine()</code>	输出一个换行字符
<code>flush()</code>	输出缓冲区里的数据
<code>close()</code>	关闭输出流
<code>clearBuffer()</code>	清除缓冲区里的数据，并将数据输出到客户端
<code>clear()</code>	清除缓冲区里的数据，但不把数据输出到客户端
<code>getBufferSize()</code>	获取缓冲区的大小
<code>getRemaining()</code>	获取缓冲区中没有被占用的空间的大小
<code>isAutoFlush()</code>	如果 <code>AutoFlush</code> 为真，则返回 <code>true</code> ，否则返回 <code>false</code>

out 对象是 JSP 中使用最频繁的对象，其 `print` 与 `println` 方法的使用更加常见。需要指出的是，`println` 方法除了把内容输出到客户端，还会在后面输出一个空行，但是该空行会在被浏览器解析时忽略（因为浏览器换行使用“`
`”）。

下面给出了使用 out 内建对象的程序的完整代码（`UseOutObject.jsp`）：

```
<%@ page contentType="text/html; charset=gb2312" %>
<html>
  <head>
    <title>使用 out 内建对象</title>
  </head>
  <body>
    <h3>使用 out 内建对象的方法进行相应操作</h3>
    <%
      out.println("<br>输出字符串：");
      out.println("字符串");
      out.println("<br>输出布尔值：");
      out.println(true);
      out.println("<br>输出 float 型数值：");
      out.println(2356.2f);
      out.println("<br>输出日期：");
      out.println(new java.util.Date().toLocaleString());
      out.println("<br>输出缓冲区大小：");
      out.println(out.getBufferSize());
      out.println("<br>输出缓冲区剩余大小：");
    %>
```

```

        out.println(out.getRemaining());
        out.println("<br>是否自动刷新缓冲区? ");
        out.println(out.isAutoFlush());
    }>
</body>
</html>

```

将 UseOutObject.jsp 文件保存到 %TOMCAT_HOME%\webapps\ROOT 目录中, 启动 Tomcat (如果此时 Tomcat 没有启动), 然后在浏览器的地址栏中输入如下内容:

<http://localhost:8877/UseOutObject.jsp>

此时浏览器中内容如图 15-22 所示。

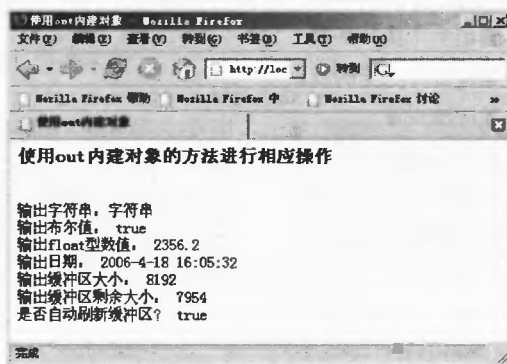


图 15-22 请求使用 out 内建对象的 JSP 界面

15.8.2 请求内建对象——request

request 对象代表请求, 该对象是 HttpServletRequest 类型的。来自客户端的请求经过容器处理以后, 由 request 对象进行封装。request 内建对象也提供了很多方法供开发人员使用, 表 15-4 给出了一些常用的方法及其说明。

表 15-4 request 内建对象的常用方法及说明

方 法	说 明
getAttribute(String name)	返回由 name 指定的属性值, 如果属性值不存在, 则返回 null
getAttributeNames()	返回 request 对象的所有属性的名字集合, 其结果为枚举对象
getCharacterEncoding()	返回请求的字符编码方式
getContentLength()	返回请求的 body 的长度, 如果长度不确定, 则返回 false
getHeader(String name)	获取 HTTP 协议定义的文件头的信息
getHeaders(String name)	返回指定名字的 request Header 的所有值, 其结果是枚举对象
getHeaderNames()	返回所有 request Header 的名字, 其结果是枚举对象
getInputStream()	返回请求的输入流, 用于获取请求中的数据
getMethod()	获取客户端向服务器端传送数据的方式, 如 get, post, header 等
getParameter(String name)	获取客户端传送给服务器的参数值, 该参数是由 name 指定的, 通常为表单中的参数
getParameterNames()	获取客户端传递给服务器的所有参数的名字, 其结果为枚举对象

续表

方 法	说 明
getParameterValues(String name)	获取指定参数的所有值
getProtocol()	获取客户端向服务器端传送参数时所依据的协议名称
getQueryString()	获取查询字符串, 该字符串是由客户端以 get 方法向服务器端传送的
getRequestURI()	获取发出请求的字符串的客户端地址
getRemoteAddr()	获取客户端的 IP 地址
getRemoteHost()	获取客户端的名字
getSession()	返回和请求相关的 session
getServerName()	获取服务器的名字
getServerPath()	获取客户端所请求的文件的途径
removeAttribute(String name)	删除请求中的一个属性
setAttribute(String name, Object)	设置名称为 name 的 request 参数的值, 该值由 Object 参数决定

下面给出了使用 request 内建对象的程序的完整代码 (UseRequestObject.jsp):

```
<%@ page contentType="text/html; charset=gb2312" %>
<html>
  <head>
    <title>使用 request 内建对象</title>
  </head>
  <body>
    <h3>使用 request 内建对象的方法进行相应操作</h3>
    <%
      out.println("<br>请求方式: ");
      out.println(request.getMethod());
      out.println("<br>获取 name 参数: ");
      out.println(request.getParameter("name"));
      out.println("<br>字符编码方式: ");
      out.println(request.getCharacterEncoding());
      out.println("<br>请求内容大小: ");
      out.println(request.getContentLength());
      out.println("<br>请求内容类型: ");
      out.println(request.getContentType());
      out.println("<br>请求协议名称: ");
      out.println(request.getProtocol());
      out.println("<br>客户端地址: ");
      out.println(request.getRequestURI());
      out.println("<br>服务器端名字: ");
      out.println(request.getServerName());
      out.println("<br>服务器端口号码: ");
      out.println(request.getServerPort());
    %>
  </body>
</html>
```

在 showUseRequest.html 中填写要提交的数据, 并将其提交到 UseRequestObject.jsp, 代码如下:

```
<html>
<head>
<title>请求 UseRequestObject</title>
</head>
<body>
<form action="UseRequestObject.jsp" name="myform" method="post">
  姓名: <input type="text" name="name"><br>
  班级: <input type="text" name="grade"><br>
  学号: <input type="text" name="stunum"><br>
  <input type="submit" value=" 确定 " >
</form>
</body>
</html>
```

将 UseRequestObject.jsp 与 showUseRequest.html 文件保存到 %TOMCAT_HOME%\webapps\ROOT 目录中, 启动 Tomcat (如果此时 Tomcat 没有启动), 然后在浏览器的地址栏中输入如下内容:

http://localhost:8877/showUseRequest.html

此时浏览器中内容如图 15-23 所示。依次在“姓名”、“班级”、“学号”文本框中填入“terrywang”、“4”和“13050506”后, 单击“确定”按钮, 此时浏览器中内容如图 15-24 所示。

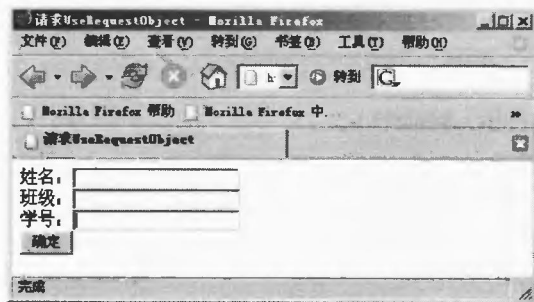


图 15-23 showUseRequest.html 界面

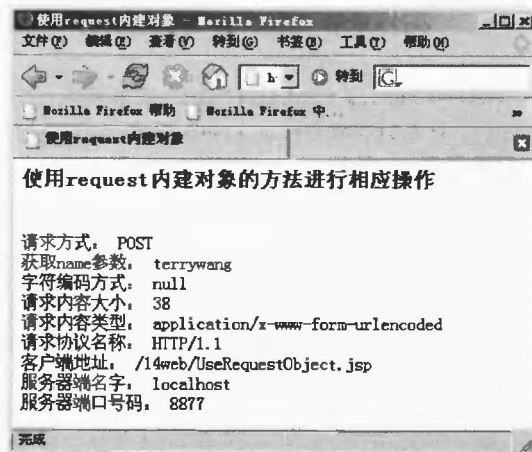


图 15-24 请求使用 request 内建对象的 JSP 界面

15.8.3 响应内建对象——response

response 是 HttpServletResponse 类的对象, 其用来封装 JSP 产生的响应, 然后把响应发送到客户端以响应客户的请求。和 request 对象一样, 它也是由容器生成。response 内建对象提供了一些方法供开发人员使用, 表 15-5 给出了其中常用的一些方法及说明。

使用 response 内建对象, 可以很方便地实现页面的重定向。在下一节中将结合 session 内建对象, 介绍如何使用 response 实现重定向。

表 15-5 response 内建对象的常用方法及说明

方 法	说 明
addHeader(String name,String value)	添加 HTTP 文件头信息, 将 Header 传到客户端去。如果已经存在同名的 Header, 则会将其覆盖
containsHeader(String name)	判断指定名称的 Header 是否存在
encoderURL()	使用 session 对象来封装 URL
flushBuffer()	强制把当前缓冲区的内容发送到客户端
getBufferSize()	返回缓冲区的大小
getOutputStream()	返回到客户端的输出流对象
sendError(int error)	向客户端发送错误信息, 如 404 是指网页不存在或者当前请求的页面无效
sendRedirect(String location)	把响应发送到另一个位置进行处理
setContentType(String ContentType)	设置响应的 MIMI 类型
setHeader(String name,String value)	设置指定名称的 Http 头文件的值, 如果该值已经存在, 则新值会覆盖原有的值

15.8.4 会话内建对象——session

session 为 javax.servlet.http.HttpSession 型对象, 该内建对象主要用来保存用户会话信息, 以跟踪每个用户的操作状态。

session 的信息保存在容器里。一般情况下用户首次登录系统时, 容器会给此用户分配一个惟一标识的 session ID, 该 ID 用于与其他用户进行区分。当用户退出系统时, 该 session ID 便会自动消失。session 内建对象提供了很多方法供开发人员使用, 表 15-6 给出了一些常用的方法及其说明。

表 15-6 session 内建对象的常用方法及说明

方 法	说 明
getAttribute(String name)	获取与指定参数相关联的属性
getAttributeNames()	返回 session 对象中存储的每一个属性对象, 其结果为枚举对象
getID()	返回一个 session ID, 每一个 ID 都是不同的
getCreationTime()	返回 session 被创建的时间, 单位为毫秒
getLastAccessedTime()	返回和当前 session 对象相关的客户端最后发送请求的时间, 单位为毫秒
getMaxInactiveInterval()	返回超时时间, 负值表示永远不会超时
invalidate()	销毁该 session 对象, 使得与其绑定的对象都失效
isNew()	如果客户端不接受使用 session, 那么每一个请求中都会产生一个 session 对象
removeAttribute(String name)	删除与指定 name 属性相关联的 session
setAttribute(String name,Object value)	设置指定名字的 name 的属性值, 并将其存储在 session 对象中

session 对象和客户端的会话紧密联系在一起, 由容器自动创建。下面给出了使用 session 的例子, 在一个页面中填写登录信息, 如果登录成功, 则把信息保存在 session 中, 在另一个页面中显示这些信息。如果登录不成功, 仍返回到登录页面。

在 UserLogin.html 中填写要提交的数据, 并将其提交到 CheckUserLogin.jsp。UserLogin.html 的代码如下:


```

<html>
  <head>
    <title>用户登录</title>
  </head>
  <body>
    <form action="CheckUserLogin.jsp" name="myform" method="post">
      用户名: <input type="text" name="name"><br>
      密码: <input type="text" name="password"><br>
      登录类型: <input type="radio" name="type" value="student">学生
               <input type="radio" name="type" value="teacher">老师<br>
               <input type="submit" value=" 确定 " >
    </form>
  </body>
</html>

```

CheckUserLogin.jsp 用来对登录信息进行验证, 并根据验证结果转发到不同的页面, 其中代码如下所示。

```

<%@ page contentType="text/html; charset=gb2312" %>
<html>
  <head>
    <title>使用 forward 动作指令</title>
  </head>
  <body>
    <%
      String name=request.getParameter("name");
      String password=request.getParameter("password");
      String logintype=request.getParameter("type");
      if((name!=null&&(name.equals("terrywang"))
        &&(password!=null&&(password.equals("123456")))){
        //如果成功登录, 则设置 name 与 type 属性, 并转到 SuccessLogin.jsp,
        //否则转到 UserLogin.html 页面
        session.setAttribute("name",name);
        session.setAttribute("type",logintype);
        response.sendRedirect("SuccessLogin.jsp"); //使用 sendRedirect
        //实现重定向
      }else{
        response.sendRedirect("UserLogin.html");
      }
    %>
  </body>
</html>

```

提示

网页重定向和转发不同, 转发不但指向新的地址而且会把原来的请求中的内容也发给新的地址, 而重定向只是指向新的地址, 不会转发请求。

SuccessLogin.jsp 用来从 session 中获取信息并输出, 代码如下:

```

<%@ page contentType="text/html; charset=gb2312" %>

```

```
<h3>以下是您的登录信息:</h3>
```

```
用户名:<%=session.getAttribute("name")%><br>
```

```
登录类型:<%=session.getAttribute("type")%>
```

将 UserLogin.html、CheckUserLogin.jsp 和 SuccessLogin.jsp 文件保存到 %TOMCAT_HOME%\webapps\ROOT 目录中,启动 Tomcat (如果此时 Tomcat 没有启动),然后在浏览器的地址栏中输入如下内容:

```
http://localhost:8877/UserLogin.html
```

此时浏览器中内容如图 15-25 所示。

在“用户名”与“密码”文本框中填入“terrywang”与“123456”,“登录类型”为“学生”,单击“确定”按钮提交表单,此时浏览器中内容如图 15-26 所示。

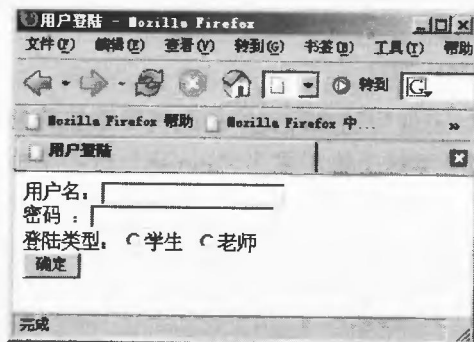


图 15-25 UserLogin.html 页面

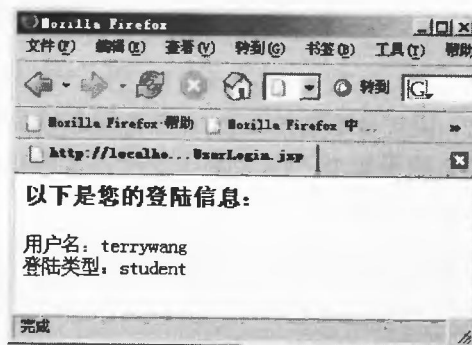


图 15-26 使用 session 内建对象

从运行中可以看出由于把用户登录的信息存储在 Session 中,当重定向到其他网页时登录的信息依然存在。

15.9 在 NetBeans 中开发完整的 JSP 程序

在本章前面几节中,介绍了与 JSP 有关的一些知识。本节将在 NetBeans 中开发一个完整的 JSP 程序,主要包括如下内容:

- 添加 HTML 文件;
- 向 HTML 页面中添加控件;
- 开发处理表单数据的 JSP。

15.9.1 添加 HTML 文件

在本节中,将在 NetBeans 中开发一个 HTML 页面,并在后面的小节中开发一个用来处理该 HTML 页面中数据的 JSP 程序,进一步介绍如何在 NetBeans 中开发 JSP 程序。

NetBeans 提供了一种半可视化的开发环境来开发 Web 程序。可以从 Palette 面板中向编辑器中拖放控件,以可视化的方式设置属性及其值。最后,该控件在编辑器中以代码的方式显示出来。

打开前面创建的项目 DemoJspApplication,按照如下步骤完成该例。

(1) 向项目中添加一个 HTML 文件：在项目窗口中选中“Web Pages”节点，单击鼠标右键，在弹出的菜单中依次选择“New”/“HTML”选项，出现如图 15-27 所示窗口。

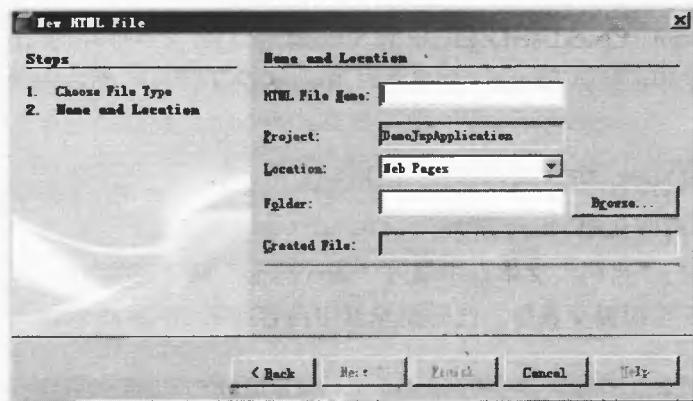


图 15-27 添加 HTML 窗口

(2) 设置要添加的 HTML 文件的名称、位置以及使用的文件夹。在此将要添加的 HTML 文件的名称设置为“UserMessages”，其他选项均使用默认值。单击“Finish”按钮完成。此时编辑器中内容如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title></title>
  </head>
  <body>
  </body>
</html>
```

此时就已经成功在 NetBeans 中添加了一个 HTML 文件。

15.9.2 向 HTML 页面中添加控件

上一节介绍了如何在 NetBeans 中创建 HTML 文件，本节将介绍如何向 HTML 文件文件中添加控件。按照如下步骤向 HTML 页面中添加控件。

(1) 向 UserMessages 中添加一个 Form：在 Palette 面板中单击鼠标左键选中“Form”选项，拖动鼠标并在编辑器中的适当位置释放鼠标左键（释放鼠标的位置即为添加的 Form 的 HTML 代码位置），出现如图 15-28 所示窗口。

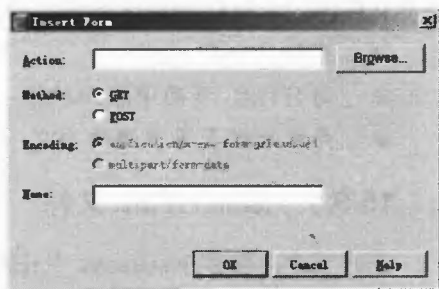



图 15-28 添加 Form 窗口

提示 可以在 Action 文本框中设置该 Form 中数据要提交到的 URL 地址，还可以设置提交表单的方式：GET 或者 POST，另外还可以设置编码方式以及该表单的名称。

(2) 将 Action 的值设置为 DealUserMessages.jsp，提交方式设置为 POST，使用默认的编

码方式, 并将 Name 的值设置为 “UserMessagesForm”。单击 “OK” 按钮完成。此时编辑器中会出现如下代码:

```
<form name="UserMessagesForm" action="DealUserMessages.jsp" method="POST">
</form>
```

 **说明** DealUserMessages.jsp 的说明将在下一个小节中给出, 它主要用来处理表单中的数据。

(3) 向 UserMessagesForm 表单中添加一个 Text Input: 在 Palette 面板中选中 “Text Input” 选项, 拖动鼠标并在编辑器中 <form> 与 </form> 标记之间置释放鼠标左键, 出现如图 15-29 所示窗口。

 **说明** 释放鼠标的位置即为添加的 Text Input 的位置。

(4) 将 “Name” 的值设置为 “userName”, 该值即为添加的 Text Input 的名称, Initial Value 的值设置为空。将 “Type” 的值设置为 “text”, 该值即为添加的 Text Input 的类型。 “Initial State” 用来设置该 Text Input 的初始状态, 这里使用默认值。 “width” 用来设置该 Text Input 的宽度, 这里设置为 30。单击 “OK” 按钮完成。

(5) 向 UserMessagesForm 表单中再次添加一个 Text Input, 将 Name 的值设置为 “userAddr”, 宽度设置为 30, 其他选项均使用默认值。

(6) 向 UserMessagesForm 表单中添加一个 Radio Button: 在 Palette 面板中用鼠标左键选中 “Radio Button” 选项, 拖动鼠标并在编辑器中的适当位置释放鼠标左键, 出现如图 15-30 所示窗口。

(7) 将 Group 的值设置为 “gender”, 具有相同 Group 值的单选按钮不能同时被选中。将 Value 的值设置为 “男”, 初始状态设置为 “selected”。单击 “OK” 按钮完成。

(8) 向 UserMessagesForm 表单中再次添加一个 Radio Button, 将 Group 的值设置为 gender, Value 的值设置为 “女”。初始状态设置为默认值。

(9) 向 UserMessagesForm 表单中添加一个 CheckBox: 在 Palette 面板中选中 “CheckBox” 选项, 拖动鼠标并在编辑器中的适当位置释放鼠标左键, 出现如图 15-31 所示窗口。

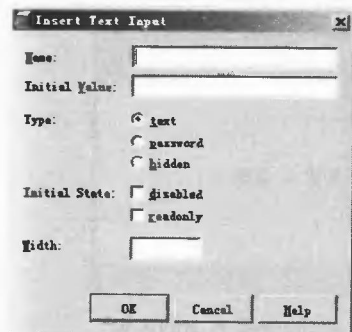


图 15-29 添加 Text Input 窗口

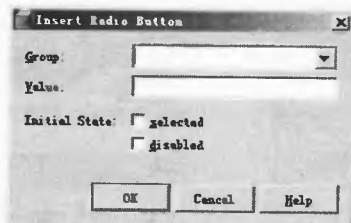


图 15-30 添加 Radio Button 窗口

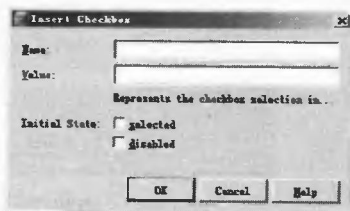


图 15-31 添加 CheckBox 窗口

(10) 将 Name 的值设置为 “favor”, 该值即为添加的 CheckBox 的名称, Value 值设置为

“唱歌”，该值即为添加的 CheckBox 的值。Initial State 使用默认值。单击“OK”按钮完成。

(11) 向 UserMessagesForm 表单中再添加 3 个 CheckBox，并按照表 15-7 所示设置其属性。

表 15-7 属性与值对照表

Name 值	Value 值	初始状态	Name 值	Value 值	初始状态
favor	跳舞	默认	favor	篮球	默认
favor	足球	默认			

(12) 向 UserMessagesForm 表单中添加一个 Button：在 Palette 面板中选中“Button”选项，拖动鼠标并在编辑器中的适当位置释放左键，出现如图 15-32 所示窗口。

(13) 将 Label 的值设置为“提交”，该值即为按钮上显示的文字。Type 值表示按钮的类型，有 3 个可选项 submit，reset 与 standard，这里使用默认值“submit”。其余选项均使用默认值。

(14) 在源代码编辑器中添加一些文本内容，这些内容用来表示相应控件中所填写内容的用途。添加完成以后，编辑器中<form>与</form>标记之间的内容如下：

```
<form name="UserMessagesForm" action=" DealUserMessages.jsp "
method="POST">
    姓名: <input type="text" name="userName" value="" width="30" /><br>
    地址: <input type="text" name="userAddr" value="" width="30" /><br>
    性别: <input type="radio" name="gender" value="男" checked />男
    <input type="radio" name="gender" value="女" />女<br>
    爱好<input type="checkbox" name="favor" value="唱歌" />唱歌
    <input type="checkbox" name="favor" value="跳舞" />跳舞
    <input type="checkbox" name="favor" value="足球" />足球
    <input type="checkbox" name="favor" value="篮球" />篮球<br>
    <input type="submit" value="提交" />
</form>
```

(15) 在项目窗口中右键单击“UserMessages.html”节点，选择菜单中的“Run File”选项，打开该 HTML 文件，如图 15-33 所示。

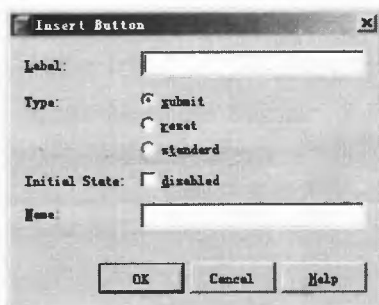


图 15-32 添加按钮窗口

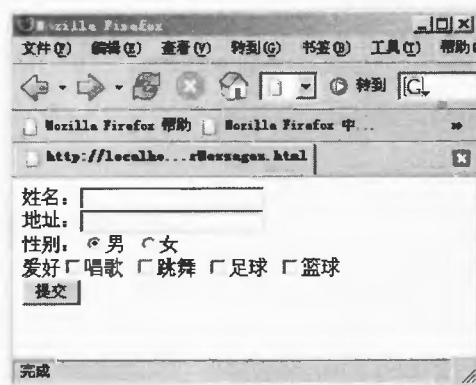


图 15-33 UserMessages.html 页面的内容

可以看到，在 NetBeans 中开发 HTML 是一件非常简单的事情，根本不需要用户编写很多的代码，只需要拖动鼠标，设置属性即可完成。

15.9.3 开发处理表单数据的 JSP 程序

本节将开发一个用来处理上节所创建的 UserMessages.html 中表单提交数据的 JSP 程序, 按照如下步骤完成该例。

- (1) 向 DemoJspApplication 中添加一个名称为 “DealUserMessages” 的 JSP 文件。
- (2) 将 title 标记的值设置为 “处理表单数据”。
- (3) 删除 body 标记内的代码, 该段代码用于用户提交的数据并输出。将下列代码添加到其中。

```
<h3>以下是用户提交的信息</h3>
<h4>
<%
    request.setCharacterEncoding("GBK");//设置请求字符编码方式
    String userName=request.getParameter("userName");//获取用户姓名
    String userAddr=request.getParameter("userAddr");//获取地址
    String gender=request.getParameter("gender");//获取性别
    String favors[] = request.getParameterValues("favor");//获取爱好的内容
    //输出从客户端获取的内容
    out.println("姓名: "+userName+"<br>");
    out.println("地址: "+userAddr+"<br>");
    out.println("性别: "+gender+"<br>");
    out.println("爱好: ");
    if(favors!=null){
        for(int i=0; i<favors.length; i++)
        {
            out.println(favors[i] + ", ");
        }
    }
%>
</h4>
```

- (4) 在项目窗口中右键单击 “UserMessages.html” 节点, 选择菜单中的 “Run File” 选项, 打开该 HTML 文件, 如图 15-33 所示。

- (5) 在图 15-33 所示窗口中输入一定内容, 单击 “提交” 按钮, 此时窗口内容如图 15-34 所示。

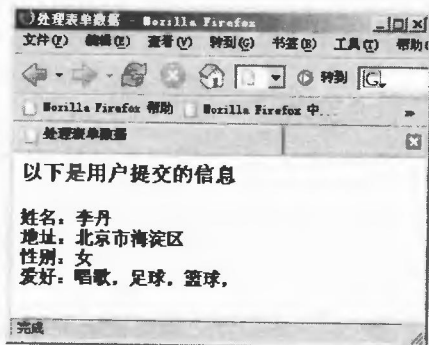


图 15-34 输出用户提交的数据

此时就已经成功的在 JSP 中接收了 HTML 页面提交的数据并进行了相应的处理。

15.10 小结

本章结合具体的实例，介绍了 JSP 中常用的各种元素，包括模板元素、指令元素、动作元素，并介绍了 JSP 中常用的几种内建对象。另外本章介绍了如何在 NetBeans 中创建使用 JSP 的项目，开发使用 JSP 的程序。在下一章中，将为读者讲述与 Servlet 相关的内容。

第 16 章

Servlet 技术及应用

上一章介绍了开发 Web 应用时经常使用的 JSP 技术，本章将会对另一种常用的 Web 技术——Servlet 进行详细的介绍。本章主要包括如下内容：

- Servlet 技术简介；
- 开发部署一个简单的 Servlet；
- 在 NetBeans 中开发 Servlet；
- 与 Servlet 有关的常用接口和类；
- 使用 HttpServlet 处理客户端请求；
- 配置 Servlet；
- 在 NetBeans 中使用 Servlet 实现文件上传。

16.1 Servlet 技术简介

Servlet 是可以在网络上响应用户请求以生成动态内容的 Java 类。可以看作是 Java 编写的类似于 CGI（公共网关接口）的一种技术，但其功能要比 CGI 强大的多。Servlet 的当前最新版本为 2.4。

Servlet 给开发人员带来了很多好处，其中最主要的是可以很容易地处理客户端传来的 HTTP 请求，并向客户端发送一个响应。Servlet 是一个 Java 类，Java 语言能够实现的功能，Servlet 基本都能实现，这就使得开发 Web 应用的能力大大增强。

一般情况下，Servlet 都作为 Web 应用程序的一部分安装在 Web 容器中。其生命周期由容器进行管理，图 16-1 显示了 Servlet 的生命周期。

下面给出了 Servlet 生命周期的详细说明。

- Servlet 容器调用 Servlet 的 `init()` 方法，

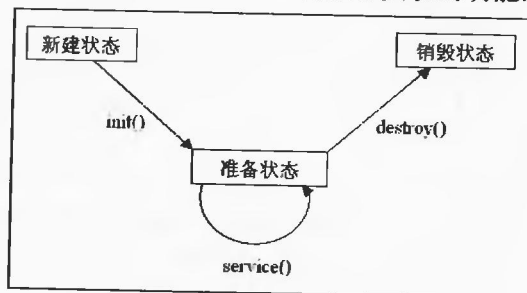


图 16-1 Servlet 的生命周期

- 用于初始化 Servlet 使用的资源。在 Servlet 的生命周期中，init 方法只被调用一次。
- init 方法初始化一个实现了 javax.servlet.ServletConfig 接口的对象。该对象为 Servlet 提供声明于部署描述文件中的初始化参数。ServletConfig 还为 Servlet 提供对 javax.servlet.ServletContext 的访问。Servlet 可以使用这个对象记录消息日志，将请求分发至其他 Web 组件，并访问同一程序中的其他 Web 资源。
- Servlet 容器调用 Servlet 的 service 方法响应请求。对于 HttpServlet，service 方法会根据不同的 HTTP 请求类型自动调用适当的方法来处理请求，如调用 doGet 方法处理 Get 请求。
- 当调用 HttpServlet 的两个主要方法 doGet 或 doPost 时，Servlet 容器会创建 javax.servlet.http.HttpServletRequest 及 HttpServletResponse 对象，并将其作为参数传递给请求处理方法。HttpServletRequest 对象用来封装请求，HttpServletResponse 对象用来封装面向此请求的 Servlet 响应。
- 当 Servlet 容器将要从服务器中移除 Servlet 时，调用 Servlet 的 destroy 方法，此时 Servlet 可以释放数据库连接等资源。

在 Servlet 的生命周期中，service 是用户最关心的方法，其中包含业务处理代码或在其调用进行业务处理的方法。图 16-2 给出了 Servlet 为用户提供服务的过程。

如果是第一次访问 Servlet，容器还会调用 init 方法来初始化 Servlet 将要使用的资源。

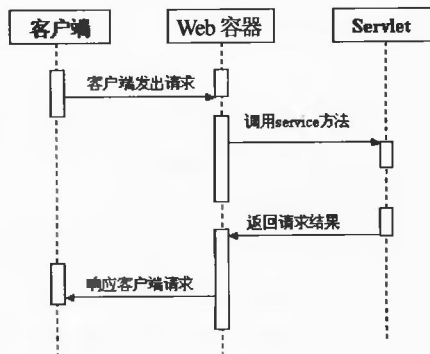


图 16-2 Servlet 为用户提供服务的过程

16.2 开发并部署一个简单的 Servlet

Hello World 几乎是学习一门新技术必开发的一个例子。本节将开发一个简单的类似于 Hello World 的 Servlet 程序并讲述如何在 Tomcat 中部署 Servlet。按照以下步骤完成该例。

(1) 新建一个名称为 HelloServlet.java 的文件，并向其中添加代码。下面给出了 HelloServlet.java 的完整代码：

```
package com.netbeans.web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet{
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                      throws ServletException, IOException{
        resp.setContentType("text/html;charset=gb2312");//设置响应类型
        PrintWriter out=resp.getWriter();//获取PrintWriter对象，用来向客户端输出内容
        out.println("<html>");
        out.println("<head>");
        out.println("<title>第一个Servlet</title>");
    }
}
```

```
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>");
        out.println("Hello Servlet!<br>");
        out.println("你好 Servlet<br>");
        out.println("当前系统时间");
        out.println(new java.util.Date().toLocaleString());
        out.println("</h2>");
        out.println("</body>");
        out.println("</html>");
    }

    public void doPost(HttpServletRequest req,
                       HttpServletResponse resp)
        throws ServletException, IOException{
        this.doGet(req, resp);
    }
}
```

对这段代码作如下说明：

- 开发的 Servlet 一般情况下都直接继承 HttpServlet，该方法封装了基于 Http 协议的 Servlet 的大部分功能，提供了 doGet 和 doPost 等众多方法。
- doGet 与 doPost 完成同样的功能，首先设置响应类型及编码方式，然后获得输出流对象并向客户端输出一些内容。

(2) 将 HelloServlet 保存在 %TOMCAT_HOME%\webapps\ROOT\WEB-INF\classes 目录中，然后编译该文件。

提示 Servlet API 位于 java 的扩展包中，如果要想成功编译上述文件，则需要将 Servlet API 的 jar 包添加到类路径中。此 jar 文件位于 %TOMCAT_HOME%\common\lib 目录中，名称为 servlet-api.jar。

(3) 打开 %TOMCAT_HOME%\webapps\ROOT\WEB-INF 目录中名称为 web.xml 的 XML 文件，将下列代码添加到该文件中的 <web-app> 与 </web-app> 标记之间。

```
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>com.netbeans.web.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/HelloServlet</url-pattern>
</servlet-mapping>
```

说明 web.xml 是该 Servlet 的配置文件。它首先使用 <servlet> 标记声明 Servlet，指定该 Servlet 的名称及其使用的类的类名全称。接着使用 <servlet-mapping> 标记为该 Servlet 进行 URL 地址映射（需要通过该映射来访问该 Servlet）。在以后的章节中会对这些内容进行详细介绍。

(4) 启动 Tomcat（如果此时 Tomcat 已经启动，则需要重新启动），在浏览器中输入如下内容：

http://localhost:8877/HelloServlet

其中“HelloServlet”为该Servlet的URL映射值。此时浏览器中内容如图16-3所示。

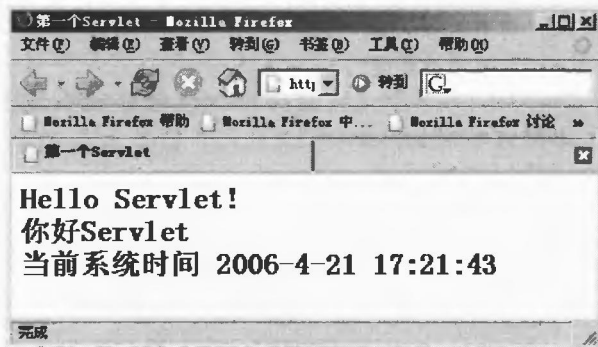


图 16-3 在浏览器中请求 Servlet

提示 如果更改了Servlet的配置文件的内容,则需要重新启动Tomcat所做的更改方会生效。

16.3 在 NetBeans 中开发与配置 Servlet

本节通过一个例子介绍了如何在NetBeans中开发及配置Servlet,在后面的几节中将会对与Servlet有关的知识进行介绍并给出一些例子。读者可以参照本节介绍的内容,在NetBeans中开发及配置这些例子,本节主要包括如下内容:

- 在NetBeans中开发Servlet;
- 在NetBeans中配置及运行Servlet。

16.3.1 在 NetBeans 中开发 Servlet

启动NetBeans,按照如下步骤完成该Servlet的开发。

- (1) 新建一个名称为DemoServletApplication的Web项目。
- (2) 在项目中创建包:在项目窗口中右键单击“DemoServletApplication”节点,在弹出的菜单中依次选择“New”/“Java Package”选项,出现如图16-4所示窗口。
- (3) 设置添加包的包名以及存储位置,内容如表16-1所示。

表 16-1 添加包的信息

字段名	说明	内容
Package Name	用来指定新添加的包名	com.netbeans.web
Location	用来指定新添加包的位置	Source Packages

(4) 单击“Finish”按钮完成包的添加。此时在项目窗口中“Source Packages”节点下会出现一个名称为“com.netbeans.web”的节点,如图16-5所示。

(5) 向com.netbeans.web包中添加Servlet:在项目窗口中右键单击“com.netbeans.web”节点,在弹出的菜单中依次选择“New”/“Servlet”选项,出现图16-6所示窗口。

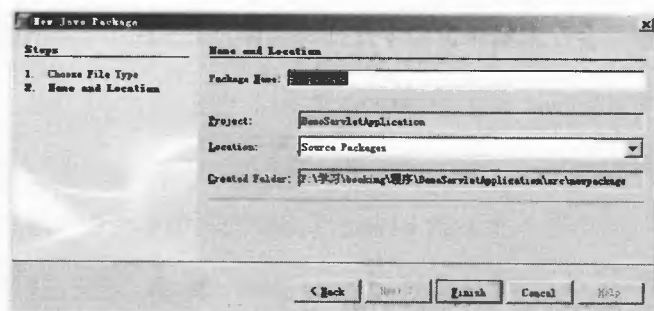


图 16-4 添加包窗口

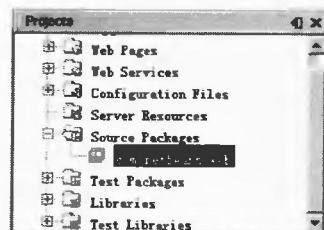


图 16-5 添加包后的项目窗口

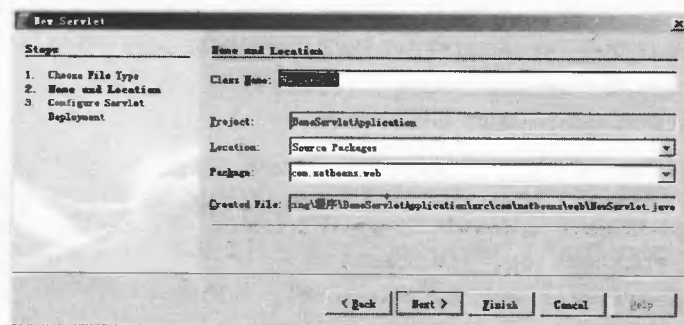


图 16-6 添加 Servlet 窗口

(6) 设置要添加的 Servlet 的类名称、位置以及使用的包。在此将 Servlet 的类名设置为 FirstServlet。单击“Next”按钮，此时窗口如图 16-7 所示。

(7) 设置该 Servlet 的名称以及使用的 URL 映射并设置初始化 Servlet 时使用的参数。对于本程序，这些选项均使用默认值。单击“Finish”按钮完成添加。

提示 如果不选中“Add information.....”复选框，该窗口中所有的选项均不可编辑。

(8) 此时，在项目窗口中“Source Package”节点的“com.netbeans.web”节点下会出现图 16-8 所示内容，表明成功的向包 com.netbeans.web 中添加了一个名称为 FirstServlet 的 Servlet。

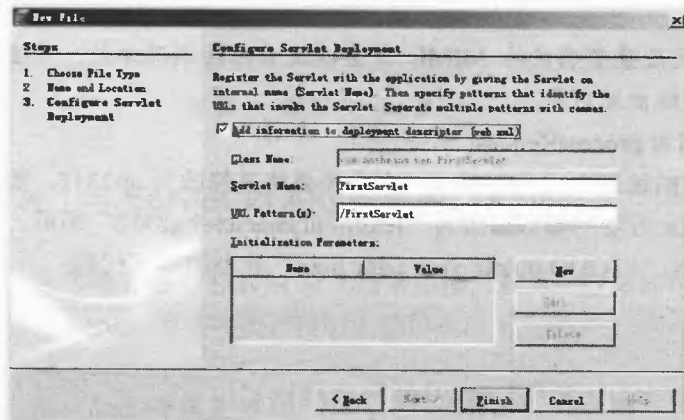


图 16-7 设置 Servlet 相关选项

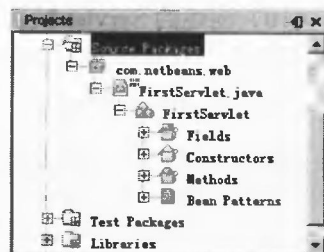


图 16-8 成功添加 Servlet

(9) 双击“FirstServlet.java”节点，会在编辑器窗口中打开该文件。下面给出了 FirstServlet.java 文件的主要内容。

```
package com.netbeans.web;
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
                                   HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        //TODO output your page here
        out.close();
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }
    public String getServletInfo() {
        return "Short description";
    }
}
```

对这段代码作如下说明。

- 可以看到，在 NetBeans 中开发 Servlet，NetBeans 自动重写了 HttpServlet 中的 doGet 方法、doPost 方法和 getServletInfo 方法。另外，NetBeans 提供了一个包含业务逻辑的方法 processRequest。
- 在 processRequest 方法内设置了响应的 MIME 类型以及响应的编码方式，并使用 response 对象获得了一个输出流对象。
- doGet 与 doPost 方法均调用 processRequest 方法。

(10) 创建的 Servlet 默认使用的编码方式为 UTF-8，在此需要将其修改为 gb2312。要修改编码方式，只需将 setContentType 方法的参数修改为“text/html;charset=gb2312”即可。

(11) 向 processRequest 方法内“//TODO output your page here”处添加如下代码：

```
out.println("<html>");
out.println("<head>");
out.println("<title>在 NetBeans 中开发 Servlet </title>");
out.println("</head>");
out.println("<body>");
```



```
out.println("<h2>这是一个 Servlet 的运行界面</h2>");  
out.println("<h2>该 Servlet 是在 NetBeans 中开发的</h2>");  
out.println("<h2>使用 NetBeans 开发 Servlet 非常方便</h2>");  
out.println("</body>");  
out.println("</html>");
```

完成上述步骤，即完成了 Servlet 的开发。

16.3.2 在 NetBeans 中配置及运行 Servlet

上一节介绍了 Servlet 的开发过程。开发出 Servlet 以后，需要对其进行相应的配置才能运行。按照如下步骤，完成对上一节开发的 Servlet 的配置。

(1) 在项目窗口中选中“Configuration Files”节点下的“web.xml”节点，双击该节点在编辑器窗口中打开 web.xml 文件。在编辑器窗口中按下“Servlets”按钮，此时编辑器窗口中内容如图 16-9 所示。

(2) 设置 Servlet 的名字、描述、Servlet 类的名称（全称类名）、URL 映射以及初始化参数等内容。在此，各个选项均使用默认设置。

(3) 单击“Pages”按钮，将欢迎文件设置为“FirstServlet”（FirstServlet 的 URL 映射名称）。

(4) 编译并运行此项目（如果没有启动 Tomcat，NetBeans 会自动启动 Tomcat），NetBeans 会自动打开默认浏览器并在浏览器中请求该 Servlet。浏览器中内容如图 16-10 所示。

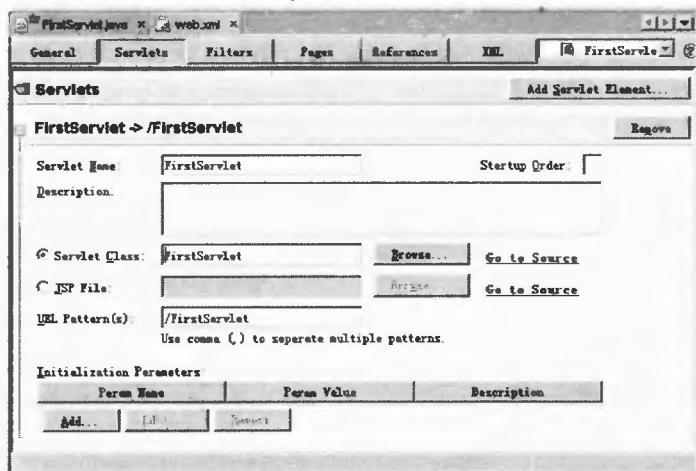


图 16-9 配置 Servlet 窗口

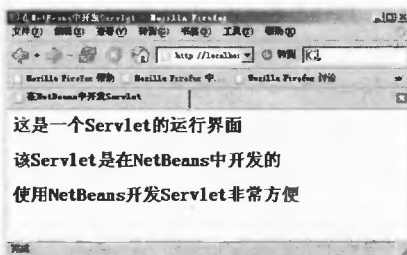


图 16-10 请求 servlet 界面

16.4 与 Servlet 有关的常用接口和类

本节将介绍与 Servlet 有关的常用接口和类，使读者对 Servlet 框架有一个更加全面的了解。与 Servlet 有关的类和接口主要包括以下几种。

- Servlet 实现相关；
- Servlet 配置相关；
- Servlet 异常相关；

- 请求与响应相关;
- 会话相关;
- Servlet 上下文;
- 请求转发器——RequestDispatcher。

16.4.1 Servlet 实现相关

与 Servlet 实现相关的接口和类主要包括 Servlet 接口、GenericServlet 类以及 HttpServlet 类。

1. Servlet 接口

javax.servlet.Servlet 接口是所有的 Servlet 都必须直接或间接实现的接口,其中定义了如下方法。

- init(ServletConfig config): 该方法用来初始化 Servlet。
- destroy(): 该方法用来销毁 Servlet。
- getServletInfo(): 获取 Servlet 的信息。
- getServletConfig(): 获取 Servlet 的配置相关信息。
- service(ServletRequest req, ServletResponse res): 应用程序逻辑的入口点,其使用两个参数: ServletRequest 表示客户端的请求, ServletResponse 表示对客户端的响应。

2. GenericServlet 类

javax.servlet.GenericServlet 是一个抽象类,提供了对 Servlet 接口的基本实现。其中最主要的 service 方法是一个抽象方法,要扩展 GenericServlet 类,必须直接或间接地实现该方法。

3. HttpServlet 类

javax.servlet.http.HttpServlet 扩展了 javax.servlet.GenericServlet, 也是一个抽象类。HttpServlet 类是针对使用 Http 协议的 Web 服务器的 Servlet 类。要扩展该类,需要实现以下方法中的一个或多个。

- doGet(): 处理 Http Get 请求的方法。
- doPost(): 处理 Http Post 请求的方法。
- doPut(): 处理 Http Put 请求的方法。
- doDelete(): 处理 Http Delete 请求的方法。
- init(): 初始化 Servlet 所需的资源。
- destroy(): 销毁 Servlet 占用的资源。
- getServletInfo(): 获取 Servlet 自身的信息。

HttpServlet 是开发 Servlet 时非常重要的类,一般情况下都会通过扩展该类来开发 Servlet,在后面的章节中,会对该类进行更加详细地介绍。

16.4.2 Servlet 配置相关

javax.servlet.ServletConfig 接口定义了与 Servlet 配置有关的内容。Servlet 配置包括 Servlet 的名字、Servlet 类的名字以及初始化参数等内容。以下代码说明了一个典型的 Servlet 配置。

```

<servlet>
  <servlet-name>HelloServlet</servlet-name>
  <servlet-class>com.netbeans.web.HelloServlet</servlet-class>
  <init-param>
    <param-name>param</param-name>
    <param-value>100</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url-pattern>/helloservlet</url-pattern>
</servlet-mapping>

```

ServletConfig 接口中定义了大量的方法供开发人员使用，表 16-2 给出了其中常用的方法及说明。

表 16-2 ServletConfig 的常用方法及说明

方 法	说 明
getContext(java.lang.String uripath)	返回当前服务器上由参数 uripath 指定的一个 ServletContext 对象
getMajorVersion()	返回当前服务器所支持的 Servlet API 的主要版本号，如当前服务器支持的版本号码为 2.3，则该方法返回 2
getMinorVersion()	返回当前服务器所支持的 Servlet API 的次要版本号，如当前服务器支持的版本号码为 2.3，则该方法返回 3
getMimeType(java.lang.String file)	返回指定文件的 MIME 类型，如果其 MIME 类型是未知的，则返回 null。
getResourcePaths(java.lang.String path)	返回当前 Web 应用程序中包含与参数 path 相同的符号的所有目录和文件列表
getRequestDispatcher(java.lang.String path)	根据给定的路径返回一个 RequestDispatcher 对象
getNamedDispatcher(java.lang.String name)	根据给定的名称返回一个 RequestDispatcher 对象
log(java.lang.String msg)	向 Servlet 的日志文件中写入信息
getRealPath(java.lang.String path)	返回给定路径在 Web 容器中的真实路径
getServerInfo()	返回当前运行的 servlet 容器的名称和版本号
getInitParameter(java.lang.String name)	返回指定名称的参数的初始值
getInitParameterNames()	返回容器中所有初始化参数的名字，返回值为一个枚举类型对象
getAttribute(java.lang.String name)	返回使用由参数 name 指定的属性，如果没有该属性，返回 null
getAttributeNames()	返回容器中所有属性的名称，返回值为一个枚举类型的对象
setAttribute(java.lang.String name, java.lang.Object object)	使用指定的值 object 设置由 name 指定的属性值
removeAttribute(java.lang.String name)	从容器中移除由 name 标识的属性
getServletContextName()	返回当前 Web 应用程序的 ServletContext 的名字

配置 Servlet 是一项比较复杂的工作，在本章后面的章节中将会详细的讲述如何配置 Servlet。

16.4.3 Servlet 异常相关

与 Servlet 异常有关的类主要包括 ServletException 与 UnavailableException。ServletException

扩展了 `java.lang.Exception`，其主要包括如下几个构造器：

- `ServletException()`;
- `ServletException(java.lang.String message)`;
- `ServletException(java.lang.String message, java.lang.Throwable rootCause)`;
- `ServletException(java.lang.Throwable rootCause)`。

`UnavailableException` 扩展了 `ServletException` 类，当 `Servlet` 或 `Filter` 暂时（或永久）不能使用时，则会产生该异常。

16.4.4 请求与响应相关

与请求和响应相关的类和接口非常多，主要包括如下几种。

- `ServletRequest`：表示 `Servlet` 的请求，是一个抽象层次比较高的接口，`HttpServletRequest` 是其子接口。
- `ServletResponse`：表示 `Servlet` 的响应，是一个抽象层次比较高的接口，`HttpServletResponse` 是其子接口。
- `ServletInputStream`：表示 `Servlet` 的输入流。
- `ServletOutputStream`：表示 `Servlet` 的输出流。
- `ServletRequestWrapper`：该类是 `ServletRequest` 接口的实现类。
- `ServletResponseWrapper`：该类是 `ServletResponse` 接口的实现类。
- `HttpServletRequest`：表示 `Http` 请求，继承了 `ServletRequest` 接口。
- `HttpServletResponse`：表示 `Http` 请求，继承了 `ServletResponse` 接口。
- `HttpServletRequestWrapper`：`HttpServletRequest` 的实现。
- `HttpServletResponseWrapper`：`HttpServletResponse` 的实现。

在上面列出的与请求相关的类和接口中，最主要的是 `HttpServletRequest` 和 `HttpServletResponse` 接口。在下面的两个小节中，将会详细介绍这两个接口。

16.4.5 `HttpServletRequest` 接口

`HttpServletRequest` 接口经常被用来获得请求中的参数，这些参数一般是客户端提交的表单中的数据。使用 `HttpServletRequest` 接口中的方法还可以获取客户端发送的数据的其他内容，主要包括：

- 客户端发送的参数名称；
- 客户端正在使用的通信协议；
- 获取服务器的主机名称；
- 获取接收请求服务器的远端主机 IP 地址；
- 获取客户端向服务器端传送数据的方式。

`HttpServletRequest` 中提供了大量的方法供开发人员使用，表 16-3 给出了其中常用的一些方法及说明。

表 16-3 `HttpServletRequest` 中的常用方法及说明

方 法	说 明
<code>getAttribute(String name)</code>	返回由 <code>name</code> 指定的属性值，如果属性值不存在，则返回 <code>null</code>
<code>getAttributeNames()</code>	返回 <code>request</code> 对象的所有属性的名字集合，其结果为枚举对象

续表

方 法	说 明
<code>getCharacterEncoding()</code>	返回请求的字符编码方式
<code>getContentLength()</code>	返回请求的 body 的长度, 如果长度不确定, 则返回 false
<code>getHeader(String name)</code>	获取 HTTP 协议定义的文件头的信息
<code>getHeaders(String name)</code>	返回指定名字的 request Header 的所有值, 其结果是枚举对象
<code>getHeaderNames()</code>	返回所有 request Header 的名字, 其结果是枚举对象
<code>getInputStream()</code>	返回请求的输入流, 用于获取请求中的数据
<code>getMethod()</code>	获取客户端向服务器端传送数据的方式, 如 get、post、header 等
<code>getParameter(String name)</code>	获取客户端传送给服务器的参数值, 该参数是由 name 指定的, 通常为表单中的参数
<code>getParameterNames()</code>	获取客户端传递给服务器的所有参数的名字, 其结果为枚举对象
<code>getParameterValues(String name)</code>	获取指定参数的所有值
<code>getProtocol()</code>	获取客户端向服务器端传送参数时所依据的协议名称
<code>getQueryString()</code>	获取查询字符串, 该字符串是由客户端以 get 方法向服务器端传送的
<code>getRequestURI()</code>	获取发出请求的字符串的客户端地址
<code>getRemoteAddr()</code>	获取客户端的 IP 地址
<code>getRemoteHost()</code>	获取客户端的名字
<code>getSession()</code>	返回和请求相关的 session
<code>getServerName()</code>	获取服务器的名字
<code>getServerPath()</code>	获取客户端所请求的文件的路径
<code>removeAttribute(String name)</code>	删除请求中的一个属性
<code>setAttribute(String name, Object)</code>	设置名称为 name 的 request 参数的值, 该值由 Object 参数决定

下面给出了一个使用 `HttpServletRequest` 的示例项目, 按照如下步骤完成开发。

(1) 开发用于从客户端获取信息的 Servlet——`GetRequest`, 下面给出了该 Servlet 的完整代码 (`GetRequest.java`):

```
package com.netbeans.web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetRequest extends HttpServlet{
    public void doGet(HttpServletRequest req,
                       HttpServletResponse resp)
        throws ServletException, IOException{
        resp.setContentType("text/html;charset=gb2312");
        PrintWriter out=resp.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>从客户端获取数据</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>以下是从客户端获取的数据: </h2>");
    }
}
```

```

        //获取并输出用户名
        out.println("<h3>用户名:" + req.getParameter("username") + "<br>");
        //获取并输出密码
        out.println("密码:" + req.getParameter("password"));
        out.println("</h3>");
        out.println("</body>");
        out.println("</html>");
    }
    public void doPost(HttpServletRequest req,
                       HttpServletResponse resp)
        throws ServletException, IOException {
        this.doGet(req, resp);
    }
}

```

(2) 将 `GetRequest` 保存在 `%TOMCAT_HOME%\webapps\ROOT\WEB-INF\classes` 目录后, 编译该文件。

(3) 编写向 `GetRequest` 提交数据的 HTML 文件——`SendRequest.html`, `SendRequest.html` 中代码如下:

```

<html>
<head>
<title>该页面用来向表单中提交数据</title>
</head>
<body>
<form action="GetRequest" name="myform" method="post">
    用户名: <input type="text" name="username"><br>
    密码: <input type="text" name="password"><br>
    <input type="submit" value="确定">
</form>
</body>
</html>

```

(4) 将 `SendRequest.html` 保存在 `%TOMCAT_HOME%\webapps\ROOT` 目录中。

提示 action 的值 “`GetRequest`” 即为要接收该 HTML 请求的 Servlet 的 URL 映射名称。

(5) 将如下代码添加到 `web.xml` 文件中, 用来对 Servlet 进行配置。

```

<servlet>
    <servlet-name>GetRequest</servlet-name>
    <servlet-class>com.netbeans.web.GetRequest</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>GetRequest</servlet-name>
    <url-pattern>/GetRequest</url-pattern>
</servlet-mapping>

```

(6) 启动 Tomcat (如果已经启动了 Tomcat, 则需重新启动)。在浏览器的地址栏中输入如下内容:

```
http://localhost:8877/SendRequest.html
```

(7) 此时浏览器中内容如图 16-11 所示。

(8) 在“用户名”与“密码”文本框中分别输入“terrywang”与“123456”, 单击“确定”按钮提交, 此时浏览器中内容如图 16-12 所示。

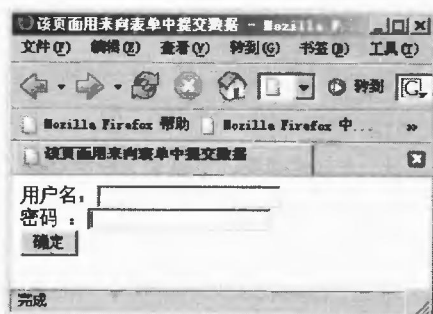


图 16-11 请求 SendRequest.html 页面

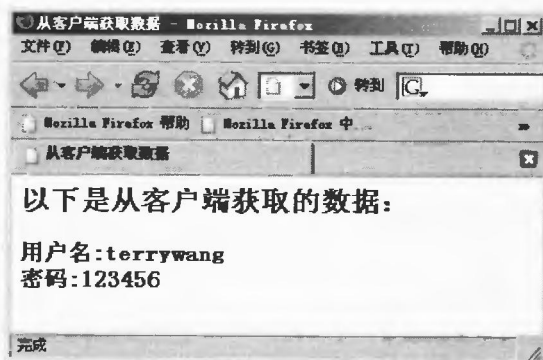


图 16-12 使用 HttpServletRequest 获取客户端数据

16.4.6 HttpServletResponse 接口

HttpServletResponse 表示服务器对客户端 Http 请求的响应。可以使用 HttpServletResponse 接口设置内容长度和响应的 MIME 类型, 并且可以提供输出流 ServletOutputStream。表 16-4 给出了 HttpServletResponse 接口中的常用方法及说明。

表 16-4 HttpServletResponse 接口的常用方法及说明

方 法	说 明
addHeader(String name,String value)	添加 HTTP 文件头信息, 该 Header 将传到客户端去。如果已经存在同名的 Header, 则会将其覆盖
containsHeader(String name)	判断指定名称的 Header 是否存在
encoderURL()	使用 session 对象来封装 URL
flushBuffer()	强制把当前缓冲区的内容发送到客户端
getBufferSize()	返回缓冲区的大小
getOutputStream()	返回到客户端的输出流对象
sendError(int error)	向客户端发送错误信息, 如 404 是只网页不存在或者当前请求的页面无效
sendRedirect(String location)	把响应发送到另一个位置进行处理
setContentType(String ContentType)	设置响应的 MIME 类型

使用 HttpServletResponse 最多的是设置响应的类型, 以下代码给出了如何使用 HttpServletResponse 设置响应类型。

```
resp.setContentType("text/html;charset=gb2312"); //resp 是 HttpServletResponse 对象
```


16.4.7 会话相关

与会话相关的类或接口主要是 `javax.servlet.http.HttpSession` 接口。

`HttpSession` 被 `Servlet` 用来实现 `Http` 客户端和 `Http` 会话两者之间的关联。这种关联可能在处理连接和请求中持续一段给定的时间。`HttpSession` 对象用来在无状态的 `Http` 协议下越过多个页面来维持状态和识别用户。

`HttpSession` 接口提供了一些方法供开发人员使用,表 16-5 给出了其中一些常用的方法及说明。

表 16-5 `HttpSession` 接口中的常用方法及说明

方 法	说 明
<code>getAttribute(String name)</code>	获取与指定参数相关联的属性
<code>getAttributeNames()</code>	返回 <code>session</code> 对象中存储的每一个属性对象,其结果为枚举对象
<code>getID()</code>	返回一个 <code>session ID</code> ,每一个 <code>ID</code> 都是不同的
<code>getCreationTime()</code>	返回 <code>session</code> 被创建的时间,单位为毫秒
<code>getLastAccessedTime()</code>	返回和当前 <code>session</code> 对象相关的客户端最后发送请求的时间,单位为毫秒
<code>getMaxInactiveInterval()</code>	返回超时时间,负值表示永远不会超时
<code>invalidate()</code>	销毁该 <code>session</code> 对象,使得与其绑定的对象都失效
<code>isNew()</code>	如果客户端不接受使用 <code>session</code> ,那么每一个请求中都会产生一个 <code>session</code> 对象
<code>removeAttribute(String name)</code>	删除与指定 <code>name</code> 属性相关联的 <code>session</code>
<code>setAttribute(String name,Object value)</code>	设置指定名字的 <code>name</code> 的属性值,并将其存储在 <code>session</code> 对象中

`HttpSession` 的一个重要应用就是可以在不同的页面之间共享数据。下面给出了使用 `HttpSession` 的 `Servlet` 的完整代码 (`UseHttpSession.java`)。在下一节中将介绍用来添加信息的 `HTML` 页面以及获取这些信息的 `JSP` 页面。

```
package com.netbeans.web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class UseHttpSession extends HttpServlet{
    public void doGet(HttpServletRequest req,
                       HttpServletResponse resp)
        throws ServletException,IOException{
        resp.setContentType("text/html;charset=gb2312");
        req.setCharacterEncoding("gb2312");
        PrintWriter out=resp.getWriter();
        //使用 HttpServletRequest 对象从客户端获取数据
        String name=req.getParameter("name");
        String age=req.getParameter("age");
        String phonenum=req.getParameter("phonenum");
        String address=req.getParameter("address");
        String othermessage=req.getParameter("othermessage");
```



```

//获取 HttpSession 对象
HttpSession session=req.getSession();
//用获取的值设置 session 属性
session.setAttribute("sesname",name);
session.setAttribute("sesage",age);
session.setAttribute("sesphonenum",phonenum);
session.setAttribute("sesaddress",address);
session.setAttribute("sesothermessage",othermessage);
out.println("<html>");
out.println("<head>");
out.println("<title>使用 HttpSession</title>");
out.println("</head>");
out.println("<body>");
out.println("<h3>已经将数据保存在 HttpSession 中</h3>");
out.println("<h3>可以在其他页面中访问这些数据</h3>");
out.println("</body>");
out.println("</html>");
}
public void doPost(HttpServletRequest req,
                    HttpServletResponse resp)
                    throws ServletException,IOException{
    this.doGet(req,resp);
}
}

```

将该文件保存在%TOMCAT_HOME%\webapps\ROOT\WEB-INF\classes 目录中,编译该文件。然后将下列内容添加到 web.xml 文件中,对此 Servlet 进行配置。

```

<servlet>
    <servlet-name>UseHttpSession</servlet-name>
    <servlet-class>com.netbeans.web.UseHttpSession</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>UseHttpSession</servlet-name>
    <url-pattern>/UseHttpSession</url-pattern>
</servlet-mapping>

```

16.4.8 开发用来提交与处理信息的页面

上一节开发的 UseHttpSession 用来将用户提交的信息保存在 HttpSession 对象中。本节将开发用来填写及处理信息的页面,按照如下步骤完成。

(1) 开发供用户填写信息的 HTML 页面——createinfo.html, 其中代码如下:

```

<html>
    <head>
        <title>该页面用来向表单中提交数据</title>
    </head>

```

```
<body>
    <form action="UseHttpSession" name="myform" method="post">
        姓名: <input type="text" name="name"><br>
        年龄: <input type="text" name="age"><br>
        住址: <input type="text" name=" address "><br>
        电话号码: <input type="text" name="phonenum"><br>
        备注信息: <br>
        <textarea name="othermessage" rows="4" cols="30"></textarea><br>
        <input type="submit" value=" 确定 " >
    </form>
</body>
</html>
```

提示 action 的值 “UseHttpSession” 为要接收该 HTML 请求的 Servlet 的 URL 映射值。

(2) 将 createinfo.html 文件保存在 %TOMCAT_HOME%\webapps\ROOT 目录中。

(3) 开发用于从 HttpSession 中读取用户信息的 HTML 页面——ShowSessionInfo.jsp, 该文件将获取的用户信息输出到客户端, 该 JSP 文件的代码如下。

```
<%@ page contentType="text/html; charset=gb2312" %>
<h2>以下是从 session 中获取的信息</h2>
<h3><%
//获取 sesname 属性的值
String name=(String)session.getAttribute("sesname");
//获取 sesage 属性的值
String age=(String)session.getAttribute("sesage");
//获取 sesphonenum 属性的值
String phonenum=(String)session.getAttribute("sesphonenum");
//获取 sesaddress 属性的值
String address=(String)session.getAttribute("sesaddress");
//获取 sesothermessage 属性的值
String othermessage=(String)session.getAttribute("sesothermessage");
out.println("<br>姓名:" + name);
out.println("<br>年龄:" + age);
out.println("<br>电话:" + phonenum);
out.println("<br>联系地址:" + address);
out.println("<br>备注信息:" + othermessage);
%></h3>
```

(4) 将 ShowSessionInfo.jsp 文件保存在 %TOMCAT_HOME%\webapps\ROOT 目录中。

(5) 启动 Tomcat (如果已经启动了 Tomcat, 则需重新启动)。在浏览器的地址栏中输入如下内容:

```
http://localhost:8877/createinfo.html
```

(6) 此时浏览器中内容如图 16-13 所示。在窗口中填入相应的数据后, 单击“确定”按钮提交, 此时浏览器中内容如图 16-14 所示。

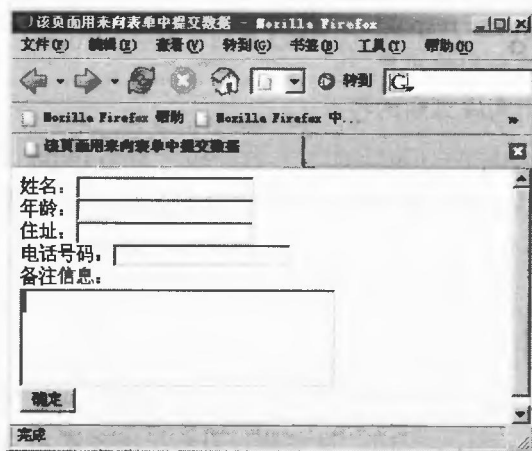


图 16-13 请求 createinfo.html 页面



图 16-14 已经保存数据

(7) 在浏览器的地址栏中输入如下内容:

```
http://localhost:8877/ShowSessionInfo.jsp
```

(8) 此时浏览器中内容如图 16-15 所示。

从图 16-15 中可以看到, ShowSessionInfo.jsp 已经成功读取了在 createinfo.html 中提交的的数据, 该数据已经由 UseHttpSession 添加到了 HttpSession 中。

16.4.9 Servlet 上下文

与 Servlet 上下文相关的类或接口是 javax.servlet.ServletContext 接口。服务器使用 HttpSession 对象来维持单个用户的状态, 而当为多个用户的 Web 应用维持一个状态时, 则应该使用 ServletContext。

ServletContext 对象表示一组 Servlet 共享的资源, ServletContext 对象提供了访问服务器名称, MIME 类型以及将信息写入服务器日志的方法。大部分的 Web 容器都会为一台主机中的所有 Servlet 或每个虚拟主机提供一个 ServletContext。

ServletContext 中常用的方法有以下几种:

- getAttribute(String name): 获取 ServletContext 中名称为 name 的属性;
- getContext(String uripath): 返回由 uripath 指定的 Servlet 上下文;
- removeAttribute(String name): 删除名称为 name 的属性;
- setAttribute(String name, Object object): 在 ServletContext 中设置一个属性, 该属性的名称由 name 指定, 值由 object 指定。

16.4.10 RequestDispatcher 接口

RequestDispatcher 主要用来进行请求转发, 其中有两个方法。

- forward(ServletRequest request, ServletResponse response): 把请求转发到同一个 Web 应用中的另一个资源 (Servlet、JSP 或者 HTML)。

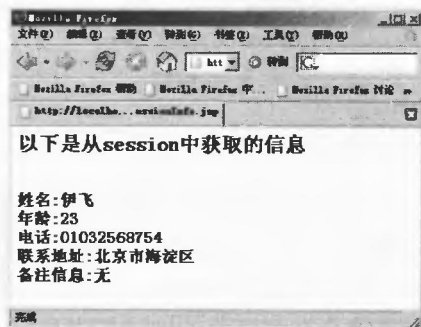


图 16-15 从 session 中获取信息

- `include(ServletRequest request, ServletResponse response)`: 将同一个 Web 应用中的另一个资源 (Servlet, JSP 或者 HTML) 包含到响应中。

在 Servlet 中, 经常使用如下代码在 Servlet 中实现页面的转发:

```
this.getServletContext().getRequestDispatcher("uripath").forward(req, resp);
```

`uripath` 参数指定了要转发到的资源的 URL, `req` 和 `resp` 分别为 `ServletRequest` 和 `ServletResponse` 对象。

16.5 使用 HttpServlet 处理客户端请求

Servlet 被设计成请求驱动。Servlet 的请求可能包含多个数据项。当 Web 容器接收到某个对 Servlet 的请求时, 容器会将请求封装为一个 `HttpServlet` 对象, 然后将该对象传递给 Servlet 中对应的服务方法。

服务方法通常是指 `doGet` 与 `doPost` 方法。另外 `HttpServlet` 还提供了其他一些方法, 如 `doPut`, `doTrace` 与 `doDelete` 方法。本节将对这些方法进行介绍, 主要包括如下内容:

- `doGet` 方法介绍;
- `doPost` 方法介绍;
- 其他方法介绍。

16.5.1 doGet 方法介绍

`doGet` 方法用来响应浏览器使用 `get` 方法提交数据的请求。使用 `get` 方法提交数据会在浏览器的地址栏中显示正在传递给 Servlet 的参数及其值, 这在系统安全方面可能会带来一些问题: 比如说用户登录时, 表单里的用户名和密码要发送到服务器端进行验证, 如果使用 `get` 方法就会在浏览器的地址栏中显示用户名和密码。

虽然可以在 `form` 中通过将 `method` 属性的值设置为 `get` 来将提交数据的方法设置为 `get`,
提示 但一般情况下都不会这么做。因为 `get` 方法不但容易泄露信息而且不能用来传送大量数据, 如文件上传。

`get` 方法的典型应用是在超链接中提交一些数据量比较小的非敏感数据。下面给出了一个该应用的示例项目, 按如下步骤完成开发:

(1) 开发使用 `get` 方法向 Servlet 提交数据的 HTML 页面——`showgetmethod.html`, 在该页面中使用超链接向 Servlet 提交了两个参数 `username` 和 `userage`, 这两个参数的值分别为 `terrywang` 与 `23`。`showgetmethod.html` 中的代码如下。

```
<html>
  <head>
    <title>使用 get 方法提交数据</title>
  </head>
  <body>
    <a href="UseDoGetMethod?username=terrywang&userage=23">
      使用 get 方法向 Servlet 提交数据</a>
    </body>
</html>
```

 **说明** href 中的值 UseDoGetMethod 为该 HTML 中数据要提交到的 Servlet 的 URL 映射。

(2) 将 showgetmethod.html 保存在 %TOMCAT_HOME%\webapps\ROOT 目录中。

(3) 开发用于处理表单中提交的数据的 Servlet——UseDoGetMethod, 下面给出了该 Servlet 的完整代码 (UseDoGetMethod.java)。

```
package com.netbeans.web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class UseHttpSession extends HttpServlet{
    public void doGet(HttpServletRequest req,
                        HttpServletResponse resp)
                        throws ServletException, IOException{
        req.setCharacterEncoding("gb2312");
        resp.setContentType("text/html;charset=gb2312");
        PrintWriter out=resp.getWriter();
        //获取使用 get 方法提交的参数 username 的值
        String name=req.getParameter("username");
        //获取使用 get 方法提交的参数 userage 的值
        String userage=req.getParameter("userage");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>使用 doGet 方法</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h3>以下是使用 doGet 方法从客户端获取的数据</h3>");
        out.println("<h3>用户名: "+name+"</h3>");
        out.println("<h3>年龄: "+userage+"</h3>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

(4) 将 UseDoGetMethod 保存在 %TOMCAT_HOME%\webapps\ROOT \WEB-INF\classes 目录后, 编译该文件。

(5) 将下列代码添加到 web.xml 文件中, 以对此 Servlet 进行配置。

```
<servlet>
    <servlet-name>UseDoGetMethod</servlet-name>
    <servlet-class>com.netbeans.web.UseDoGetMethod</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>UseDoGetMethod</servlet-name>
    <url-pattern>/UseDoGetMethod</url-pattern>
</servlet-mapping>
```


(6) 启动 Tomcat (如果已经启动了 Tomcat, 则需重新启动)。在浏览器的地址栏中输入如下内容:

`http://localhost:8877/showgetmethod.html`

(7) 此时浏览器中内容如图 16-16 所示。

(8) 单击图 16-16 中的超链接文字, 此时浏览器中内容如图 16-17 所示。



图 16-16 请求 showgetmethod.html 页面

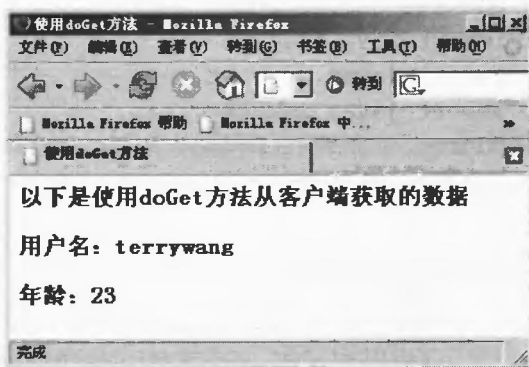


图 16-17 使用 doGet 响应请求界面

此时图 16-17 中浏览器的地址栏的内容如下:

`http://localhost:8877/UseDoGetMethod?username=terrywang&userage=23`

可以看到对于使用 get 方法提交的数据, 这些数据会在浏览器的地址栏中显示出来。

16.5.2 doPost 方法介绍

doPost 方法用于接收客户端以 post 方法发送数据的请求, 使用其可以隐藏客户端发送给服务器数据的具体内容, 提高了安全性。另外, 使用 post 方法可以发送大量的数据 (如上传文件), 而 get 方法则不可以发送大量数据。

下面给出了一个使用 doPost 方法处理客户端提交数据的示例项目, 按照如下步骤完成开发。

(1) 开发使用 post 方法发送数据的静态页面——showpostmethod.html, 其代码如下:

```
<html>
<head>
  <title>使用 post 方法发送请求</title>
</head>
<body>
  <form action="UseDoPostMethod" name="myform" method="post">
    <h2>请在下面输入要提交的数据</h2>
    <textarea name="message" rows="4" cols="30"></textarea><br>
    <input type="submit" value="提交" >
  </form>
</body>
</html>
```

(2) 将 showpostmethod.html 文件保存在 %TOMCAT_HOME%\webapps\ROOT 目录中。

下面给出了 UseDoPostMethod 的完整代码 (UseDoPostMethod.java):

```
package com.netbeans.web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class UseDoPostMethod extends HttpServlet{
    public void doPost(HttpServletRequest req,
                        HttpServletResponse resp)
                        throws ServletException, IOException{
        req.setCharacterEncoding("gb2312");
        resp.setContentType("text/html;charset=gb2312");
        PrintWriter out=resp.getWriter();
        String message=req.getParameter("message");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>使用 doPost 方法</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h3>以下是使用 doPost 方法从客户端获取的数据</h3>");
        out.println("<textarea name=message rows=4 cols=30>");
        out.println(message);
        out.println("</textarea>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

(3) 将 UseDoPostMethod 保存在 %TOMCAT_HOME%\webapps\ROOT\WEB-INF\classes 目录后, 编译该文件。

(4) 将下列代码添加 web.xml 文件中, 对此 Servlet 进行配置。

```
<servlet>
    <servlet-name>UseDoPostMethod</servlet-name>
    <servlet-class>com.netbeans.web.UseDoPostMethod</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>UseDoPostMethod</servlet-name>
    <url-pattern>/UseDoPostMethod</url-pattern>
</servlet-mapping>
```

(5) 启动 Tomcat (如果已经启动了 Tomcat, 则需重新启动), 在浏览器的地址栏中输入如下内容:

```
http://localhost:8877/showpostmethod.html
```

(6) 此时浏览器中内容如图 16-18 所示。

(7) 在图 16-18 所示页面中输入一定内容, 单击“提交”按钮提交数据, 此时浏览器中内容如图 16-19 所示。

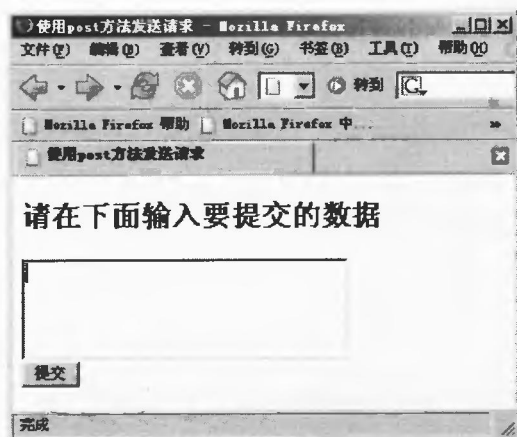


图 16-18 请求 showpostmethod.html 页面

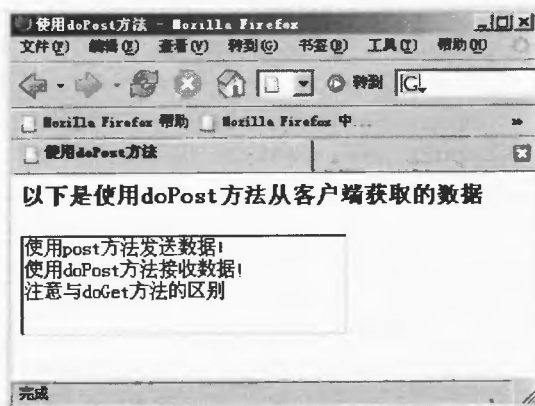


图 16-19 使用 doPost 方法处理数据

查看此时浏览器地址栏中内容，如下所示：

`http://localhost:8877/UseDoPostMethod`

可以看到使用 post 方法提交的数据，其内容并不会在浏览器的地址栏中显示出来。

16.5.3 其他方法介绍

除了上面介绍的两个常用方法之外，HttpServlet 还提供了另外几个方法用于接收客户端以不同方式提交的数据，主要包括如下几种。

- doPut: doPut 方法与 doPost 方法类似，其允许客户端把真正的文件存储在服务器上，而不仅仅是向服务器传送数据。
- doDelete: 与 doPut 方法类似，其允许客户端删除服务器端的文件或者 Web 页面。不过这种方法使用非常少。
- doTrace: 该方法由容器调用以处理 Trace 请求。该方法主要用于调试，不可以重写。
- doHeader: 该方法用于处理客户端的 Head 请求，并返回一个 response。当客户端仅仅关心响应的 Header 时，就可以发送一个 Header 请求。这种情况下客户端往往关心响应的长度和响应的 MIME 类型。
- doOptions: 该方法用于处理客户端的 Options 请求，通过该方法客户端可以获得此 Servlet 支持的方法。

16.6 配置 Servlet

前面已经陆续介绍过一些 Servlet 的配置方法，本节将总结前面介绍的配置并介绍一些新的配置选项。

总的来说，Servlet 的配置主要包括配置 Servlet 的名称、Servlet 的类（如果是 JSP，则指定 JSP 文件）、初始化参数、启动与加载的优先级以及 Servlet 的映射等内容。本节将会结合一个例子来详细地介绍这些配置，主要包括如下内容：

- 演示程序；
- Servlet 的名称、类以及其他选项；

- Servlet 的初始化参数;
- 启动和加载优先级;
- Servlet 的映射。

16.6.1 演示程序

本小节给出了一个 Servlet 示例程序,后面的几节将会根据该例介绍如何对 Servlet 进行相应的配置。下面给出了该例的完整代码 (CounterServlet.java):

```
package com.netbeans.web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CounterServlet extends HttpServlet{
    int counter=0;//用来记录此 servlet 启动后被访问的次数
    int initnum=0;//用来存放初始值
    public void doPost(HttpServletRequest req,
                       HttpServletResponse resp)
        throws ServletException,IOException{
        resp.setContentType("text/html;charset=gb2312");
        PrintWriter out=resp.getWriter();
        counter++;
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet 的配置</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h3> initnum 的初始值是: "+initnum+"</h3>");
        out.println("<h3>服务器启动以来,该 Servlet 又被访问了: ");
        out.println(counter+"次</h3>");
        out.println("</body>");
        out.println("</html>");
    }
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
        throws ServletException,IOException{
        this.doPost(req,resp);
    }
    public void init() throws ServletException{
        String initnum=this.getInitParameter("initnum");//获取 initnum 的初始值
        try{
            this.initnum=Integer.parseInt(initnum);
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

16.6.2 Servlet 的名称、类以及其他选项

在配置 Servlet 时, 首先必须要指定 Servlet 的名称、Servlet 的类 (如果是 JSP, 则指定 JSP 文件的名称), 另外还可以有一些可选的配置, 如指定其在部署时使用的名称, 以及给其增加一定的描述。

CounterServlet 的配置在 web.xml 配置文件中如下:

```
<servlet>
    <servlet-name>CounterServlet</servlet-name>
    <servlet-class>com.netbeans.web.CounterServlet</servlet-class>
    <description>学习 Servlet 的配置 </description>
    <display-name>ConfigCounterServlet</display-name>
    .....
</servlet>
```

在这段代码中, <servlet-name>为此 servlet 的名称。<servlet-class>为此 servlet 对应的类。<description>为此 servlet 的描述。<display-name>为此 servlet 的显示名称, 在管理中使用。

如果要指定的 Servlet 是一个 JSP 文件, 则可以进行如下设定。

```
<servlet>
    <servlet-name>UseJsp</servlet-name>
    <jsp-file>newUseJsp.jsp</jsp-file>
</servlet>
```

16.6.3 Servlet 的初始化参数

可以在配置文件中指定 Servlet 初始化参数的名称和数值, CounterServlet 的初始化参数设置如下:

```
<servlet>
    ...
    <init-param>
        <param-name> initnum</param-name>
        <param-value>100</param-value>
    </init-param>
</servlet>
```

在上面代码中, <param-name>指定了参数的名称, <param-value>指定了参数的值。而在 CounterServlet 中, 使用如下代码获取 initnum 参数的值:

```
String initnum =this.getInitParameter("initnum");
```

理论上可以在 Servlet 中的任何方法内获取相应属性的初始值。但在实际使用中经常在

提示 Servlet 的 init 方法内获取该值, 这样可以保证在 Servlet 的生命周期内, 该值仅被获取一次。在实际应用中, 一般都会在初始化参数中设置一些动态信息, 如端口号、数据源等内容。

16.6.4 启动和加载优先级

Servlet 之间还可能存在相互调用和依赖的关系, 在此种情况下, 应该在加载该 Servlet

之前, 加载其要使用的 Servlet。可以在配置 Servlet 时使用<load-on-startup>标记指定 Servlet 加载的优先级, 如下所示。

```
<servlet>
    <servlet-name>UseJsp</servlet-name>
    <jsp-file>newUseJsp.jsp</jsp-file>
    < load-on-startup>30</ load-on-startup >
</servlet>
<servlet>
    <servlet-name>CounterServlet</servlet-name>
    <servlet-class>com.netbeans.web.CounterServlet</servlet-class>
    < load-on-startup >10</ load-on-startup >
</servlet>
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>com.netbeans.web. HelloServlet</servlet-class>
    < load-on-startup >AnyTime</ load-on-startup >
</servlet>
```

UseJsp 的启动优先级为 30, CounterServlet 的启动优先级为 10, HelloServlet 可以在服务器启动后的任何时候被加载。这样设置, 可以确保 CounterServlet 总在 UseJsp 之前被加载。

16.6.5 Servlet 的映射

要想在浏览器中访问 Servlet, 必须要为已有的 Servlet 提供 URL 映射。可以给同一个 Servlet 设定一个或者多个 URL 映射, 这样可以通过不同的 URL 来访问同一个 Servlet。


以下代码是 CounterServlet 的映射配置。

```
<servlet-mapping>
    <servlet-name>CounterServlet</servlet-name>
    <url-pattern>/CounterServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>CounterServlet</servlet-name>
    <url-pattern>/counter</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>CounterServlet</servlet-name>
    <url-pattern>/counter/*</url-pattern>
</servlet-mapping>
```

Servlet 的映射由<servlet-mapping>标记来标识, 其中包括<servlet-name>与<url-pattern>子标记。<servlet-name>的值与<servlet>标记中给出的<servlet-name>的值相同。<url-pattern>表示在浏览器中请求该 Servlet 需要使用的 URL 值。

在此例中, 为 CounterServlet 指定了 3 个 URL 映射值, 分别为“/CounterServlet”、“/counter”与“/counter/*”。其中“/counter/*”表示只要映射以“/counter/”开头, 都可以访问该 Servlet。可以在浏览器的地址栏中输入如下内容来访问该 Servlet。

- `http://localhost:8877/CounterServlet;`
- `http://localhost:8877/ counter;`
- `http://localhost:8877/ counter/aaa` (对应于 `"/counter/*"` 映射);
- `http://localhost:8877/ counter/bbb` (对应于 `"/counter/*"` 映射)。

 **提示** 在现在使用的一些 Web 服务器中, 可以将 Servlet URL 映射值指定为以 `"/"` 结尾的值, 但是在最新的 Servlet 规范中, 已经不能够将 Servlet URL 映射值指定为以 `"/"` 结尾的值。

16.7 在 NetBeans 使用 Servlet 实现文件上传

要实现文件上传, 其主要思路是使用 JSP 或 Servlet 获取客户端请求的输入流, 然后从这个输入流中读取指定的文件, 最后将文件保存到指定的位置。可以使用第三方的上传组件完成文件上传功能, 在本节中将使用 SmartUpload 实现文件的上传。

16.7.1 开发实现文件上传的 Servlet

启动 NetBeans, 打开前面创建的 Web 项目 DemoServletApplication, 按如下步骤完成该 Servlet 的开发。

(1) 将包含 SmartUpload 的 jar 文件添加到类路径中 (本书光盘的对应目录中有相应的 jar 文件)。

(2) 向 `com.netbeans.web` 包中添加一个名称为 `UploadServlet`、URL 映射值为 `/UploadServlet` 的 Servlet。


(3) 在编辑器窗口中打开 `UploadServlet.java` 文件, 并向其中添加如下代码。

```
import com.jspsmart.upload.*; //导入 SmartUpload 相关包
```

(4) 向 `UploadServlet` 中添加一个名称为 `config`、类型为 `ServletConfig`、访问限制修饰符为 `private`、初始值为 `null` 的成员变量。

(5) 向 `UploadServlet` 中添加一个签名为 `final public void init(ServletConfig config) throws ServletException` 的方法, 将下列代码添加到该方法中:

```
this.config = config; //初始化 Servlet
```

 **提示** `init` 方法在启动 Web 容器的时候被调用, 并且在 Servlet 的整个生命周期中仅被调用一次。

(6) 删除 `doGet` 方法的方法体内的代码, 将下列代码添加到该方法的方法体内。

```
response.setContentType("text/html;charset=GBK"); //设置头信息
PrintWriter out = response.getWriter();
out.println("<h3>");
out.println("必须采用 post 方法上传文件!");
out.println("</h3>");
```

 **说明** 该段代码的作用是如果客户端使用 GET 方法请求该 Servlet, 则向客户端输出“必须采用 post 方法上传文件”。

(7) 删除 `doPost` 方法的方法体内的代码，并将下列代码添加到该方法的方法体内。

```
response.setContentType("text/html;charset=GBK");//设置响应的类型
PrintWriter out = response.getWriter();
out.println("<h3>");
int count=0; // 定义上传的文件个数
SmartUpload mySmartUpload = new SmartUpload();//创建 SmartUpload 对象
try {
    mySmartUpload.initialize(config,request,response);// 初始化
    mySmartUpload.upload();// 上载文件
    // 获取保存文件的路径
    String path=mySmartUpload.getRequest().getParameter("PATH");
    //使用给定的文件保存路径保存文件
    count = mySmartUpload.save(path);
    // 显示处理结果
    out.println("已经成功上传"+count+"个文件<br>"); //输出上传文件的个数
    out.println("已上传文件的保存路径为: "+path); //输出文件的保存路径
} catch (Exception e){
    out.println("无法上传文件.<br>");
    out.println("错误: " + e.toString());
}
out.println("</h3>");
}
```



说明 该段代码用来处理客户端的 POST 请求，从客户端读取数据并将其存储到客户端指定的服务器的相应目录中。

完成上述步骤以后，就已经成功地开发了一个 Servlet。

16.7.2 开发 JSP 程序

本节将开发一个 JSP 程序，用来设定要上传的文件以及文件的保存路径。按照如下步骤完成该 JSP 程序的开发。

(1) 向 `DemoServletApplication` 中添加名称为 `upload` 的 JSP 文件。

(2) 在编辑器中打开该文件，删除 NetBeans 自动生成的代码，并将以下代码添加到该 JSP 文件中。

```
<%@page contentType="text/html;charset=gb2312"%>
<html>
    <head>
        <title>选择要上传的文件</title>
    </head>
    <body>
        <form method="post" action="UploadServlet"
            enctype="multipart/form-data">
            <h3>
                设置目标文件夹位置: <input type="text" name="PATH" size="30"><br>
                请选择要上传的文件: <input type="file" name="FILE1" size="30"><br>
```



```
<input type="submit" value="上传">
</h3>
</form>
</body>
</html>
```

action 属性的值 UploadServlet, 即为该 HTML 页面数据要提交到的 Servlet 的 URL 映射的值。此段代码可以通过从“Palette”中拖放控件来快速实现。

(3) 将 DemoServletApplication 的欢迎文件设置为 upload.jsp。

(4) 编译并运行主项目, NetBeans 会自动启动 Tomcat 并在浏览器中打开 upload.jsp 页面。此时, 浏览器中内容如图 16-20 所示。

(5) 在图 16-20 将目标文件夹的位置设置为“C:\Downloads”, 该目录是服务器上的一个目录, 用来存储上传的文件。将要上传的文件设置为“D:\happy.jpg”, 该文件是客户端机器上的一个文件。设置完成以后, 单击“上传”按钮完成上传。此时浏览器中内容如图 16-21 所示。



图 16-20 请求 upload.jsp 页面

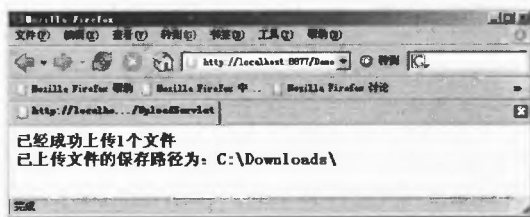


图 16-21 完成文件上传

(6) 在服务器端打开保存文件的目录 C:\Downloads\, 界面如图 16-22 所示。

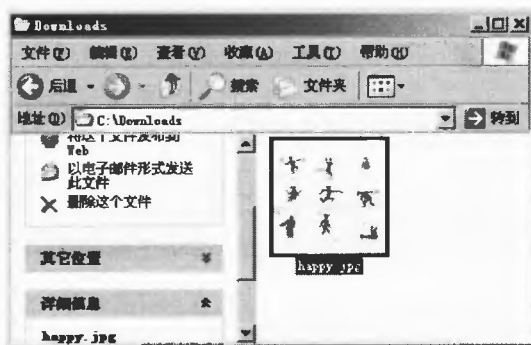


图 16-22 查看文件是否成功上传

从图 16-22 可以看到, 被上传的文件已经保存在服务器的指定目录内。

16.8 小结

本章结合具体的实例, 介绍了与 Servlet 有关的常用接口和类, 包括与请求相关的接口和类、与响应相关的接口和类以及与配置相关的接口和类等内容。介绍了如何配置 Servlet, 并对 Servlet 与 JSP 进行了对比。在本章的最后, 给出了一个使用 Servlet 实现文件上传的例子。

第 17 章

JavaBean 组件模型

JSP 最强有力的一个功能就是能够使用 JavaBean 组件模型。JavaBean 往往封装了程序的业务逻辑，是可重用的组件。本章将对与 JavaBean 组件模型的有关内容进行讨论，主要包括如下内容：

- JavaBean 简介；
- 与 JavaBean 有关的概念；
- 在 JSP 中使用 JavaBean；
- 在 NetBeans 中使用 JSP 与 JavaBean；
- 在 Servlet 中使用 JavaBean；
- HTTP 监视器。

17.1 JavaBean 简介

JavaBean 体系结构是一个全面基于组件的标准模型。JavaBean 是描述 Java 软件组件的模型，有些类似于 Microsoft 的 COM 组件。JavaBean 其实就是一个 Java 类，它遵循标准的接口规范。在规范中主要规定了方法的命名、底层行为、继承以及实现行为的标准。

早期，JavaBean 主要应用于可视化领域，如 AWT 下的应用。现在，JavaBean 更多应用在非可视化领域，它在服务器端方面的应用表现出越来越强的生命力。

非可视化的 JavaBean 与可视化的 JavaBean 使用同样的属性和事件。非可视化的 JavaBean 在 Web 应用中通常用来封装业务逻辑和数据库操作等内容，这样可以更好的实现业务逻辑和前台显示层的分离，使系统具有更好的健壮性和灵活性。

JavaBean 组件与企业级 JavaBean (Enterprise JavaBean, EJB) 组件的概念是完全不同的。EJB 是 J2EE 的核心技术，它为开发服务器端应用程序组件提供了一个模型，可以利用该模型来创建企业级应用程序或组件。有关 EJB 的内容会在下一篇中进行介绍。

17.2 与 JavaBean 有关的概念

本节将对与 JavaBean 有关的概念进行简单的介绍，主要包括如下内容：

- JavaBean 的属性简介;
- Simple 属性;
- Indexed 属性;
- Bound 属性;
- Constrained 属性;
- JavaBean 的方法。

17.2.1 JavaBean 的属性简介

属性是 JavaBean 组件内部状态的抽象表示, 实际上就是成员变量 (这些成员变量一般是 `private` 的)。JavaBean 的属性可以分为以下 4 类: Simple (简单属性)、Indexed (索引属性)、Bound (绑定属性)、Constrained (约束属性)。在接下来的几节中, 将会对这几种属性进行详细介绍。

17.2.2 简单属性

简单属性表示 JavaBean 中带有相应 `get` 或 `set` 方法的成员变量, 其 `getXXX` 或 `setXXX` 方法必须依赖于标准命名规范来定义, 使用语法如下:

```
public void set<PropertyName>(<PropertyType> value);  
public PropertyType get<PropertyName>();
```

对于 `boolean` 类型的属性, 除了可以使用 `getXXX` 方法获取属性值以外, 还可以使用 `isXXX` 方法来获取该属性值。语法如下:

```
public boolean is<PropertyName>();
```

例如, 对于 `userPassword` 这个属性, 假设该属性的属性类型是 `int`, 则可以采用下面的 `get/set` 方法设置与获取该属性的值。

```
public void setUserPassword (int userpassword);  
public int getUserPassword ();
```

可以将属性设置为可读写、只读或只写。如果该属性既具有 `getXXX` 方法又具有 `setXXX` 方法, 则该属性是可读写的; 如果该属性只具有 `getXXX` 方法, 则该属性是只读的; 如果该属性只具有 `setXXX` 方法, 则该属性是只写的。

下面给出了一个具有简单属性的 JavaBean 的完整代码 (SimpleJavaBean.java):

```
package com.netbeans.javabean;  
public class SimpleJavaBean{  
    private String userName;//声明类型为 String, 值为 userName 的属性  
    private int userPassword;//声明类型为 int, 值为 userPassword 的属性  
    private String loginType;//声明类型为 String, 值为 loginType 的属性  
    private boolean admin;//声明类型为 boolean, 值为 admin 的属性  
    public String getUserName() { //获取 userName 属性  
        return userName;  
    }  
    //设置 userName 属性
```

```
public void setUsername(String userName) {
    this.userName = userName;
}
public int getUserPassword() {
    return userPassword;
}
public void setUserPassword(int userPassword) {
    this.userPassword = userPassword;
}
public String getLoginType() {
    return loginType;
}
public void setLoginType(String loginType) {
    this.loginType = loginType;
}
//admin 为 boolean 类型, 因此可以使用 isXXX 方法获取属性
public boolean isAdmin() {
    return admin;
}
public void setAdmin(boolean admin) {
    this.admin = admin;
}
}
```

提示 注意 get/set 方法的命名规则, 在 get/set 后加上对应的属性名称, 但是属性名称的第一个字母要变成大写, 而属性本身名称的第一个字母应该为小写。

在上面的程序中, 由于 admin 属性是 boolean 类型的, 因此可以使用 isAdmin 方法来获得这个属性的值。在这里 isAdmin 方法与 getAdmin 方法是一样的, 这两个方法之间可以进行内部转换。

17.2.3 索引属性

索引属性表示一个成员数组, 同上一节给出的简单属性一样, 其 getXXX 或 setXXX 方法必须依赖于标准命名规范来定义。使用 set 或 get 方法设置与获取索引属性值的语法如下:

```
public void set<PropertyName>(int index,<PropertyType>value);
public void set<PropertyName>(<PropertyType[]>value);
public <PropertyType[]> get<PropertyName> ();
public <PropertyType> get<PropertyName> (int index);
```

下面给出了使用索引属性的完整代码 (IndexedJavaBean.java):

```
package com.netbeans.javabean;
public class IndexedJavaBean{
    //定义了一个名称为 sports 类型为 String 的属性
    private String[]sports=new String[]
        {"football","basketball","volleyball","swam","dance"};
    //返回 sports 的值
```

```
public String[] getSports(){
    return sports;
}
//设置 sports 的值
public void setSports(String[] sports){
    this.sports = sports;
}
//根据指定的索引 index 设置值 value
public void setSports(int index,String value){
    this.sports[index]=value;
}
//根据给定的索引 index 获取值
public String getSports(int index){
    return sports[index];
}
}
```

有了上面给出的 JavaBean，则可以向下面这样使用该 JavaBean：

```
String sporta=myJavaBean.getSports(3); //myJavaBean 为一个已经存在的 JavaBean 的对象
myJavaBean.setSports(2,"fighting");
myJavaBean.String [ ]sport=getSports();
```

17.2.4 绑定属性

绑定属性提供了一种监听机制，可以通过实现了 `PropertyChangeListener` 接口的监听器来监听 JavaBean 中属性值的改变。当属性值发生变化时，监听器接收由 JavaBean 组件产生的 `PropertyChangeEvent` 事件对象，`PropertyChangeEvent` 对象中封装了对应的属性名称、旧的属性值以及新的属性值。

使用绑定属性的 JavaBean 需要实现 `addPropertyChangeListener` 方法与 `removePropertyChangeListener` 方法，以便添加和移除属性变化监听器。

这些属性以及方法在 JavaBean 的图形用户界面编程中会经常用到。本章的目的是讲述如何在 Web 程序中使用 JavaBean，不再对该属性进行详细介绍，有兴趣的读者可以查阅相关资料。

17.2.5 约束属性

约束属性与绑定属性很类似，不同之处在于约束属性的值的变化首先要被所有的监听器验证之后，值的变化才能够对 JavaBean 组件产生作用。

JavaBean 组件的约束规范是指当该属性的值要发生变化时，已经与这个属性建立某种联系的其他 Java 对象可以阻止属性值的改变，并且任何一个监听器对象都可以阻止属性变化。

在这种情况下，监听器对象需要实现 `VetoableChangeListener` 接口，并接收由 JavaBean 组件产生的 `PropertyChangeEvent` 事件对象。JavaBean 组件还可以使用 `VetoableChangeSupport` 辅助程序类激活由监听器接收的实际事件。

通常情况下，使用 JavaBean 组件对象的引用来构造 `VetoableChangeSupport` 对象，并且 JavaBean 可以使用 `addVetoableChangeListener` 或 `removeVetoableChangeListener` 方法来添加或

者移除监听器。

和绑定属性一样，在 Web 程序中使用 JavaBean 时很少涉及约束属性，因此这里也不再给出详细的介绍。

17.2.6 JavaBean 的方法

JavaBean 的方法和普通的 Java 程序中的方法没有什么不同，下面为一个简单的 JavaBean 方法的代码。

```
public void showMessage() {  
    System.out.println("用户名: "+username);  
    System.out.println("密码: "+password);  
}
```

17.3 在 JSP 中使用 JavaBean

前面几节对与 JavaBean 有关的一些基础知识进行了简单的介绍。本节将会详细的介绍如何在 JSP 中使用 JavaBean，主要包括如下内容：

- <jsp:useBean>动作指令简介
- <jsp:setProperty>与<jsp:getProperty>简介

17.3.1 <jsp:useBean>动作指令简介

<jsp:useBean>指令用来在 JSP 页面中创建或获取一个 JavaBean 对象。如果该 JavaBean 对象已经存在，则获取该对象；否则就创建一个 JavaBean 对象。该指令还可以指定 JavaBean 对象的名称及作用范围，以保证对象在指定的范围内可以使用。其定义语法如下：

```
<jsp:useBean id=" beanid" scope="page | request | session | application"  
typeSpec />
```

其中 beanid 是一个大小写相关的名字，用来表示该 JavaBean 实例；scope 表示该对象的作用范围；typeSpec 可以是以下 4 者之一：

- class="classname";
- class="classname" type="typename";
- beanName="beanname" type="typename";
- type="typename"。

例如：

```
<jsp:useBean id="userbean" scope="session" class="app.UserBean" />  
<jsp:useBean id="loginbean" scope="session" class="app.LoginBean"/>
```

下面给出了<jsp:useBean>指令中各个属性的详细说明。

- id="beanid": 在所定义的范围中标识 Bean 的变量。在定义某个 JavaBean 时，需要指定其 id，通过该 id 对 JavaBean 的对象进行引用。该变量名必须符合 Java 命名规范的要求，并且变量名称对大小写敏感。
- 如果要使用的是一个已经创建好的 Bean，那么该 id 的值必须与原来 id 的值一致。
- scope="page | request | session | application": 定义 Bean 的作用范围，scope 的默认值

是 page。表 17-1 给出了这 4 种值的详细说明。

表 17-1 scope 属性值及说明

属 性	说 明
page	表示指定的 JavaBean 对象仅在当前页面中有效
request	表示指定的 JavaBean 对象在本次请求范围内有效, 其作用范围比 page 大。如果当前页包含有另外的页面 (使用了<jsp:include>), 该 JavaBean 对象在被包含的页中同样有效。同时, 如果当前页面转发了请求 (使用了<jsp:forward>), 在转发到的页面中同样有效
session	表示指定的 JavaBean 对象在会话范围内有效, 其作用范围比 page 和 request 大
application	表示指定的 JavaBean 对象在整个服务器的生命周期内都有效。服务器的生命周期对应着服务器的一次启动与关闭

- **class="classname"**: classname 为 Bean 的全称类名。该类不能是抽象的, 必须有一个公共的、没有参数的构造器。
- **BeanName="beanname" type="typename"**: 指定 Bean 的名字。如果提供了 type 属性和 beanName 属性、允许省略 class 属性。
- **type="typename"**: 指定引用该 JavaBean 对象的变量的类型, 它必须是该 JavaBean 类的名称, 超类名称或者该类所实现的接口名称之一。

提示 不能同时使用 class 和 BeanName。

17.3.2 <jsp:setProperty>与<jsp:getProperty>简介

一般情况下, <jsp:setProperty>、<jsp:getProperty>和<jsp:useBean>一起使用, 用来设置 Bean 的属性。<jsp:setProperty>指令使用 Bean 给定的 setXXX 方法在 Bean 中设置一个或者多个属性值。Bean 可以使用内省 (introspection) 机制来发现属性及其对应访问方法的名字是什么。

可以按照如下方法来使用<jsp:setProperty>。

```
<jsp:setProperty name="beanname" propertyDetails>
```

name="beanname": 该属性是必选属性, 其值为 Bean 的名字。在定义之前由<jsp:useBean>引入, 其值应该与<jsp:useBean>中 id 的值相同。下面列出了 propertyDetails 的可用值及说明。

- **property=""**: 设置 Bean 属性的快捷方式。在 Bean 中属性的名字、类型必须和 request 对象中的参数名称相匹配。由于从表单中传递过来的数据类型都是 String 类型的, JSP 内在机制会把这些参数转换成与 Bean 属性对应的类型。如果 request 对象中有空值, 则对应的 Bean 属性不会设定任何值。同样, 如果 Bean 中有一个属性没有与之对应的 request 参数值, 则该属性也不会被设定。
- **property="propertyname" param="parametername"**: 在 Bean 属性的名称和 request 中参数名称不相同时候可以使用该方法。param 指定参数名称。
- **property="propertyname"**: 使用 request 中的一个参数来指定 Bean 中的一个属性值。在该语法中, property 指定 Bean 的属性名称, 而且 Bean 属性和 request 参数的名称应该相同, 否则应该使用上面的语法指定 param。
- **property="propertyname" value="propertyvalue"**: value 是一个可选属性, 其使用指定

的值来设定 Bean 的属性。该值可以是字符串,也可以是表达式。如果是字符串,则该字符串的值会自动通过 `valueOf()` 方法包装成为与 Bean 属性一致的类型;如果是表达式,则其类型必须和将要设定的属性值的类型一致。

下面给出了使用 `<jsp:setProperty>` 的一段代码:

```
<jsp:useBean id="loginbean" class="org.LoginBean" scope="session"/>
<jsp:setProperty name="loginbean" property="password" value="abcdefg">
<jsp:setProperty name="loginbean" property="username" param="uid" />
```

对于上述这段代码,作如下说明:

- 第 2 行采用指定的值 "abcdefg" 来设置名称为 "loginbean" 的 JavaBean 中 "password" 属性的值。
- 第 3 行采用请求中的 "uid" 参数的值来设置名称为 "loginbean" 的 JavaBean 中 "username" 属性的值。



提示 不能同时在一个 `<jsp:setProperty>` 中同时使用 `param` 和 `value`, 因为两者都可设置 Bean 属性的值, 仅仅是设置方式不同。

`<jsp:getProperty>` 操作是对 `<jsp:setProperty>` 的补充, 用来获取一个 Bean 的属性值。`<jsp:getProperty>` 的使用方法如下:

```
<jsp: getProperty name="beannname" property=" propertyname" >
```

下面给出了使用 `<jsp:getProperty>` 的一段代码。

```
<jsp:useBean id="loginbean" class="org.LoginBean" scope="session"/>
<jsp: getProperty name=" loginbean" property=" username" >
```

下面给出了 `<jsp:getProperty>` 指令中各个属性的详细说明。

- `name="beannname"`: 该属性为必选属性, 其值为 Bean 的名称, 在此之前由 `<jsp:usebean>` 指令引入, 其值应该与 `<jsp:useBean>` 中 `id` 的值相同。
- `property=" propertyname"`: 该属性为必选属性, 其值为所指定 Bean 的属性的名称。

17.4 使用 JavaBean 封装数据库连接

随着 JSP 技术的不断发展和成熟, 在 JSP 页面中直接连接数据库的编程方式已经非常少见。虽然在 JSP 页面中使用 Scriptlets 也可以连接数据库, 进行数据库操作, 但这样会造成页面难以维护。在大型的项目中, 这些缺点变得更加突出。

解决这些问题的一个比较好的办法就是, 将与数据库操作相关的代码封装在 JavaBean 组件或 Servlet 组件中。这样, JavaBean 或 Servlet 组件执行后台的数据库操作并获得执行结果, 再通过 JSP 显示这些结果。

本节将要在 NetBeans 中开发一个在 JavaBean 中连接数据库的 Web 项目, 主要包括如下内容:

- 创建数据库和表;
- 配置 ODBC 数据源;

- 开发表示用户信息的 JavaBean 组件；
- 开发封装数据库操作的 JavaBean 组件；
- 开发用于提交数据的 HTML 页面；
- 开发用于注册用户信息的 JSP 页面。

17.4.1 创建数据库和表

本节开发的 JavaBean 组件涉及数据库操作，因此在开发 JavaBean 组件之前应进行相应的准备工作，包括创建数据库和表以及配置数据源。本节将为读者介绍如何创建数据库和表。在本例中使用 Access 数据库，按照如下步骤完成数据库与表的创建。

(1) 单击“开始”/“程序”/“Microsoft Access”菜单命令，弹出新建数据库对话框。在该对话框中选择“空 Access 数据库”选项后，单击“确定”按钮，弹出“文件新建数据库”对话框。

(2) 设置数据库的名称以及保存目录。此处将数据库名称设置为“userinfo”，并保存到相应目录中，此时出现如图 17-1 所示窗口。



图 17-1 userinfo 数据库

(3) 在图 17-1 所示窗口中选“使用设计器创建表”，单击鼠标右键，在弹出的菜单中选择“设计视图”选项，出现如图 17-2 所示窗口。

(4) 向“字段名称”列中添加名称为“username”的字段，并将光标移至该单元格，单击鼠标右键，在弹出的菜单中选择“主键”选项，将“username”设置为主键。

(5) 将光标移至“username”对应的“数据类型”列，单击下三角按钮，出现图 17-3 所示下拉列表框。

(6) 设置对应字段的数据类型，在此选中“文本”选项，即将“username”列的数据类型设置为文本。此时“常规”选项卡中内容如图 17-4 所示。这里给出了一些相应的设置选项，如设置字段大小、格式、是否是必填字段等内容。

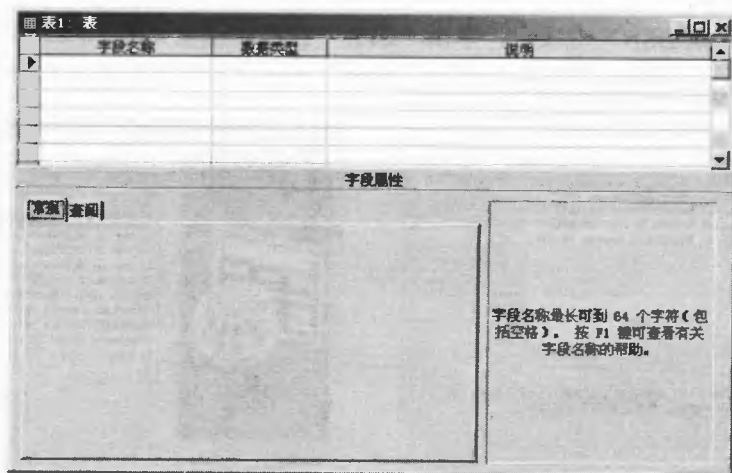


图 17-2 设计视图窗口

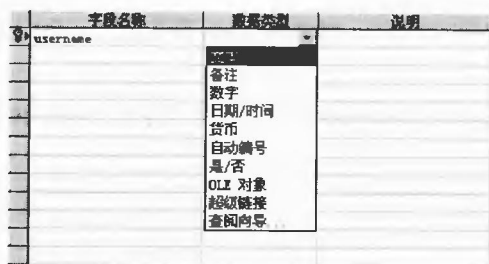


图 17-3 设置数据类型

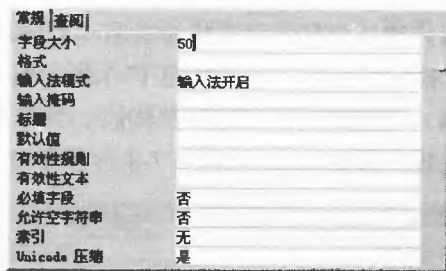


图 17-4 常规选项卡

(7) 将字段大小设置为 50, “必填字段”选项设置为“是”, 其余选项均使用默认值。

(8) 依照上述步骤, 继续向表中添加字段。表 17-2 给出了各字段的名称以及相应的设置选项。

表 17-2 添加字段的名称及相应的设置选项

字段名称	数据类型	字段大小	必填字段	允许空字符串
departname	文本	50	是	否
salary	文本	50	是	否
jobposition	文本	50	是	否
backmessages	文本	255	否	是

(9) 保存该表, 将其名称设置为 userinfo。

完成了上述步骤, 就已经成功创建了数据库及表。

17.4.2 配置 ODBC 数据源

上一节介绍了如何在 Access 中创建数据库和表, 本节将为读者介绍如何配置 ODBC 数据源。按照如下步骤完成数据源的配置。

(1) 单击“开始”/“设置”/“控制面板”命令, 在控制面板窗口中选择“管理工具”, 接着在“管理工具”中选择“数据源 (ODBC)”, 则出现如图 17-5 所示窗口。

(2) 选择“系统 DSN”选项卡，单击“添加”按钮，出现如图 17-6 所示窗口。

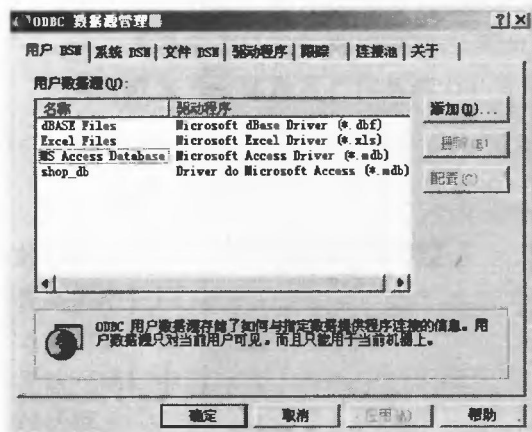


图 17-5 配置数据源窗口



图 17-6 创建数据源对话框

(3) 选择为其安装数据源的驱动程序，在此选中“Microsoft Access Driver(*.mdb)”选项，单击“确定”按钮，出现如图 17-7 所示窗口。

(4) 设置数据源名称以及其他一些相关内容。在此将数据源名称设置为“javabean”，单击“选择”按钮，出现如图 17-8 所示窗口。

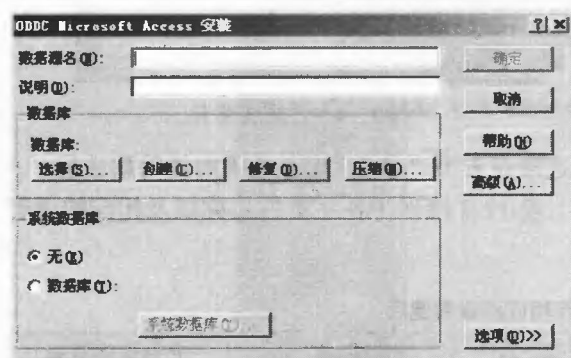


图 17-7 配置数据源相关选项

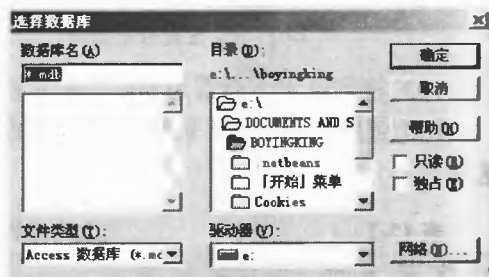


图 17-8 选择数据库

(5) 指定使用的数据库，在此将数据库指定为上一小节中创建的 userinfo，单击“确定”按钮完成。

完成了上述步骤，就成功完成了 ODBC 数据源的配置。

提示 因为 Access 没有专用的 JDBC 数据库驱动，所以给其设置了 ODBC 数据源，在后面的小节中将通过 JDBC-ODBC 桥来访问此数据库。

17.4.3 开发表示用户信息的 JavaBean 组件

本节将开发一个用来表示用户信息的 JavaBean 组件。启动 NetBeans，按照如下步骤完成该例。

- (1) 创建一个名称为 DemoJavaBeanApplication 的 Web 项目。
- (2) 向 DemoJavaBeanApplication 中添加一个名称为 com.netbeans.web 的包。

(3) 向 `com.netbeans.web` 中添加 Java 类文件: 在项目窗口中选中 `com.netbeans.web` 节点, 单击鼠标右键, 在弹出的菜单中依次选择 “New” / “Java Class” 选项, 此时窗口如图 17-9 所示。

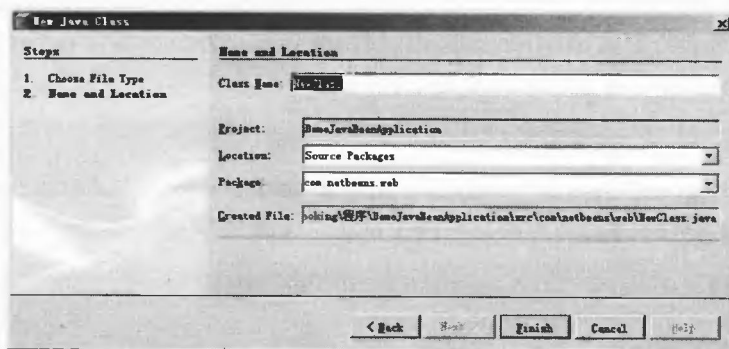


图 17-9 向包中添加 Java 类文件

(4) 设置要添加的类的名称、位置以及使用的包。在此将 “Class Name” 的值设置为 `UserInfo`, 其余选项均使用默认值。单击 “Finish” 按钮完成添加。

(5) 在编辑器中打开 `UserInfo.java`, 除去注释部分, `UserInfo.java` 中代码如下:

```
package com.netbeans.web;
public class UserInfo {
    public UserInfo() {
    }
}
```

(6) 按照表 17-3 所示向 `UserInfo` 添加 4 个成员变量。

表 17-3 添加的成员变量的属性

变量名称	类 型	访问限制修饰符	初 始 值
<code>userName</code>	<code>String</code>	<code>private</code>	<code>null</code>
<code>departName</code>	<code>String</code>	<code>private</code>	<code>null</code>
<code>salary</code>	<code>String</code>	<code>private</code>	<code>null</code>
<code>jobPosition</code>	<code>String</code>	<code>private</code>	<code>null</code>
<code>backMessages</code>	<code>String</code>	<code>private</code>	<code>null</code>

(7) 为表 17-3 中所列出的成员变量添加相应的 `get` 与 `set` 方法。

(8) 保存并编译该文件。此时就成功在 NetBeans 中开发了一个用于表示用户信息的 `JavaBean`。

17.4.4 开发封装数据库操作的 `JavaBean` 组件

前面小节开发了用于表示用户信息的 `JavaBean` 组件。在本小节中, 将开发一个用于封装数据库操作的 `JavaBean` 组件。按照如下步骤完成该 `JavaBean` 的开发。

(1) 打开上个小节中创建的项目 `DemoJavaBeanApplication`, 向包 “`com.netbeans.web`” 中添加一个名称为 `UserRegister` 的 Java 类。

(2) 按照表 17-4 所示向 `UserRegister` 中添加两个成员变量。

表 17-4

添加的成员变量的属性

变 量 名 称	类 型	访问限制修饰符号	初 始 值
con	Connection	private	null
userInfo	UserInfo	private	null

(3) 为添加的成员变量 `userInfo` 添加相应的 `set` 方法, 方法签名为 `public void setUserInfo(UserInfo userInfo)`, 并在方法体中添加如下代码:

```
this.userInfo= userInfo;
```

(4) 向 `UserRegister` 中添加一个签名为 `private void connectTODB()` 的方法, 并将以下代码添加到该方法的方法体内。

```
String CLASSFORNAME="sun.jdbc.odbc.JdbcOdbcDriver";
String CONNECTSTR="jdbc:odbc:javabeen";
try{
    Class.forName(CLASSFORNAME);
    this.con=DriverManager.getConnection(CONNECTSTR);//将获取的连接赋值给 con
} catch(Exception e) {
    e.printStackTrace();
}
```

在这段代码中, 注意以下两点。

- "jdbc:odbc:javabeen"表示通过 JDBC-ODBC 桥来连接 ODBC 数据源——javabeen 以访问数据库。
- 以上代码用来初始化数据库连接, 并将获取的数据库连接赋值给 `con`。

(5) 向 `UserRegister` 中添加一个签名为 `public void regist()` 的方法, 并将以下代码添加到该方法中:

```
String sql="insert into userinfo values(?,?,?, ?,?)";
PreparedStatement pstmt=null;//声明一个预处理对象
try{
    this.connectTODB();
    //使用 con 创建一个预处理对象, 并设置其参数
    pstmt=con.prepareStatement(sql);
    pstmt.setString(1,this.userInfo.getUserName());
    pstmt.setString(2,this.userInfo.getDepartName());
    pstmt.setString(3,this.userInfo.getSalary());
    pstmt.setString(4,this.userInfo.getJobPosition());
    pstmt.setString(5,this.userInfo.getDepartName());
    pstmt.executeUpdate();
}catch(Exception e){
    e.printStackTrace();
}finally{
    try{
        if(pstmt!=null){
            pstmt.close();
        }
    }
```

```

        if(con!=null){
            con.close();
        }
    }catch(Exception e){
        e.printStackTrace();
    }
}

```



说明

本段代码使用 con 创建了一个 PreparedStatement 对象,并设置其参数,最后使用 executeUpdate 方法将数据添加到数据库中。

通过上述步骤在 NetBeans 中开发了一个用于执行数据库操作的 JavaBean 组件。

17.4.5 开发 HTML 页面与 JSP 页面

前面两小节已经完成了 JavaBean 组件的开发,本节将开发用于提交数据的 HTML 页面与处理数据的 JSP 页面。打开前面创建的项目 DemoJavaBeanApplication,按照如下步骤完成该例。

(1) 向 DemoJavaBeanApplication 中添加一个名称为 createuserinfo.html 的页面,在编辑器中打开该文件,删除其中自动生成的代码,将以下代码添加到其中:

```

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    <title>填写用户信息界面</title>
  </head>
  <body>
    <h3>
      <form action="dealuserinfo.jsp" name="myform" method="post">
        姓名: <input type="text" name="username"><br>
        部门: <input type="text" name="departName"><br>
        薪水: <input type="text" name="salary"><br>
        职位: <input type="text" name="jobPosition"><br>
        备注: <br><textarea cols="30" rows="5" name="backMessages"></textarea><br>
          <input type="submit" value="提交">
      </form>
    </h3>
  </body>
</html>

```

(2) 向 DemoJavaBeanApplication 中添加一个名称为 dealuserinfo 的 JSP,在编辑器中打开该文件,删除其中自动生成的代码,并将下列代码添加到其中:

```

<%@page contentType="text/html; charset=gb2312"%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    <title>添加信息页面</title>
  </head>
  <body><h3>

```



```

<%
    request.setCharacterEncoding("gb2312");
%>
<jsp:useBean id="userinfo" class="com.netbeans.web.UserInfo" scope="page">
    <jsp:setProperty name="userinfo" property="*" />
</jsp:useBean>
<jsp:useBean id="userregist" class="com.netbeans.web.UserRegist" />
<%
    userregist.setUserInfo(userinfo);<!--设置 userregist 的 userInfo 属性-->
    userregist.regist();//调用 regist 方法向数据库中添加数据
    out.println("添加信息成功! <br>");
%>
添加数据库中对应用属性的值为:<br>
<!--使用 jsp:getProperty 标签获取 userinfo 中的属性并显示-->
姓名: <jsp:getProperty name="userinfo" property="userName" /><br>
部门: <jsp:getProperty name="userinfo" property="departName" /><br>
薪水: <jsp:getProperty name="userinfo" property="salary" /><br>
职位: <jsp:getProperty name="userinfo" property="jobPosition" /><br>
备注: <br><textarea cols="30" rows="4" name="backMessages">
    <jsp:getProperty name="userinfo" property="backMessages" />
</textarea><br>
</h3></body>
</html>

```

说明 以上代码使用从客户端获取的数据设置 `UserInfo` 的相应属性, 用该对象设置 `UserRegist` 中的 `userInfo` 属性。完成了向数据库中添加数据的功能, 并输出已经添加到数据库中的相应内容。

- (3) 将 `DemoJavaBeanApplication` 的欢迎页面设置为 `createuserinfo.html`。
- (4) 运行该项目, 在弹出的浏览器中输入一些相应的信息, 此时浏览器中内容如图 17-10 所示。
- (5) 单击“提交”按钮, 浏览器中内容如图 17-11 所示。

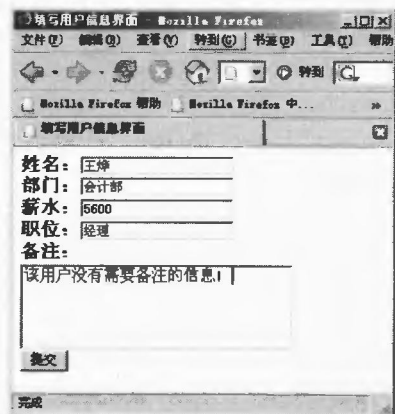


图 17-10 添加信息后的 `createuserinfo.html` 页面

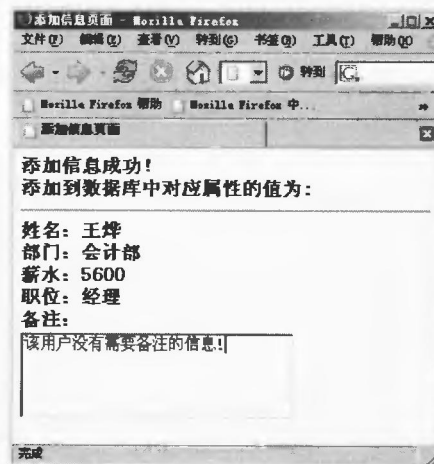


图 17-11 成功添加到数据库

此时如果查看数据库,会发现有一条相应的记录被添加到表 `userinfo` 中。通过该项目可以看到,将数据库操作封装在 `JavaBean` 中可以减少 `JSP` 中程序片段的使用,使 `JSP` 更加易于修改和控制,实现了界面与业务相分离,增加了系统的可维护性。这种方法已经越来越被开发人员所接受。

17.5 用 `JavaBean` 实现购物车

上一节介绍了如何在 `JavaBean` 中封装数据库操作,本节将开发一个简单的购物车程序,来进一步介绍 `JavaBean` 的使用。启动 `NetBeans`,按照如下步骤完成该例。

(1) 打开前面创建的名称为“`DemoJavaBeanApplication`”的 `Web` 项目。

(2) 向 `com.netbeans.web` 包中添加一个名称为 `Cart` 的 `Java` 类,该类即作为本例的 `JavaBean` 组件。

(3) 向 `Cart` 中添加一个名称为 `cartHash`、类型为 `Hashtable`、访问限制修饰符号为 `private`、初始值为 `new Hashtable()` 的成员变量。

(4) 向 `Cart` 中添加一个签名为 `public Hashtable getGoods()` 的方法,并将下列代码添加到该方法中:

```
return this.cartHash;
```

(5) 再向 `Cart` 中添加一个签名为 `public void addGoods(String goodnun)` 的方法,并将下列代码添加到该方法中:

```
String no=goodnun.trim();
if(this.cartHash.containsKey(no)){//如果当前购物车中已经存在该商品,则将该商品数量加1。
    int num=Integer.parseInt(cartHash.get(no).toString());
    cartHash.put(no,num+1);
}else{//如果当前购物车中不存在该商品,则添加该商品,并将其值设置为1。
    cartHash.put(no,1);
}
```

(6) 向 `DemoJavaBeanApplication` 中添加一个名称为 `usecart` 的 `JSP` 文件。删除该文件内自动生成的代码并将下列代码添加到其中。

```
<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
        <title>使用 JavaBean 的购物车</title>
    </head>
    <body><h4>
        选择商品后单击“提交”按钮添加到购物车
        <form action="usecart.jsp" method="POST">
            <select name="good" size="6">
                <option value="0">NetBeans 从入门到精通</option>
                <option value="1">Java 2 参考手册</option>
```

```

        <option value="2">Java 数据库开发宝典</option>
        <option value="3">J2EE 开发使用手册</option>
        <option value="4">Java 网络编程</option>
    </select><br>
    <input type="submit" value="提交">
</form>
<%! String goodName=null; %>
<jsp:useBean id="cart" class="com.netbeans.web.Cart" scope="session"/>
<%//向购物车中添加商品
    goodName=request.getParameter("good");
    if(goodName!=null)
        cart.addGoods(goodName);
%>
<%//输出购物车中商品
    Hashtable cartHash=cart.getGoods();
    Set cartSet=cartHash.keySet();
    Iterator cartIt=cartSet.iterator();
    while(cartIt.hasNext()){
        int num=Integer.parseInt(cartIt.next().toString());
        switch(num){
            case 0:
                out.println("NetBeans 从入门到精通: "+cartHash.get("0")+"本<br>");
                break;
            case 1:
                out.println("Java2 参考手册: "+cartHash.get("1")+"本<br>");
                break;
            case 2:
                out.println("Java 数据库开发宝典: "+cartHash.get("2")+"本<br>");
                break;
            case 3:
                out.println("J2EE 开发使用手册: "+cartHash.get("3")+"本<br>");
                break;
            case 4:
                out.println("Java 网络编程: "+cartHash.get("4")+"本<br>");
                break;
        }
    }
%>

```

(7) 将 usecart.jsp 设置为 DemoJavaBeanApplication 的欢迎页面后, 运行主程序, 此时浏览器中内容如图 17-12 所示。

可以在窗口的列表框中选商品, 单击“提交”按钮将商品添加到购物车中。选中

“NetBeans 从入门到精通”选项，单击“提交”按钮，此时浏览器中内容如图 17-13 所示。

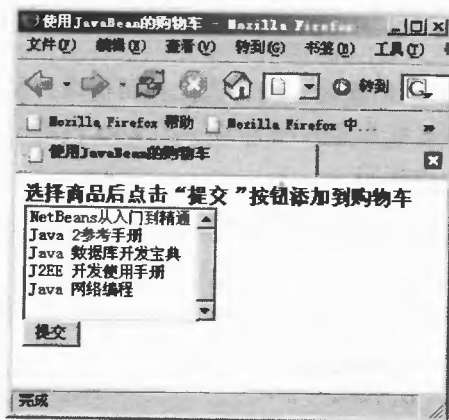


图 17-12 程序运行界面

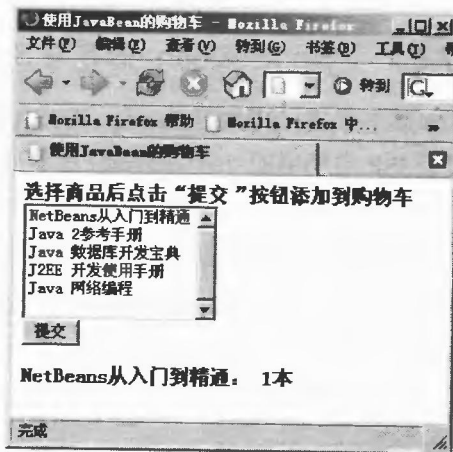


图 17-13 将商品添加到购物车

如果向购物车中添加购物车中已经存在的商品，则该商品的数量会自动加 1，如图 17-14 所示。

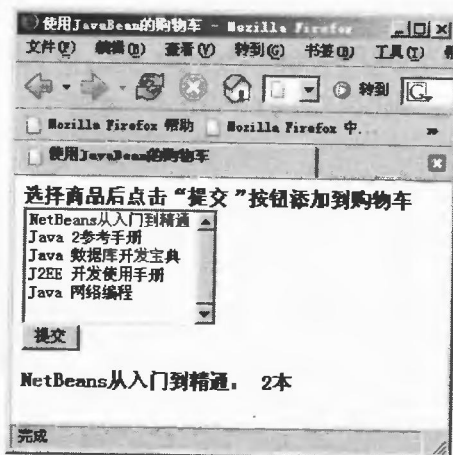


图 17-14 向购物车中添加购物车中已经存在的商品

17.6 在 Servlet 中使用 JavaBean

Servlet 没有提供类似于 JSP 中<jsp:useBean>的标签，不能使用类似<jsp:setProperty>的标签设置 JavaBean 中相应属性的值，也不能使用类似<jsp:getProperty>的标签获取 JavaBean 中相应属性的值。

本节将在 Netbeans 中开发一个在 Servlet 中使用 JavaBean 的程序，以介绍如何在 Servlet 中使用 JavaBean，主要包括如下内容：

- <jsp:useBean>标签与 Servlet;
- 开发 JavaBean;

- 开发 Servlet;
- 开发 HTML 与 JSP。

17.6.1 <jsp:useBean>标签与 Servlet

从本质上来说, JSP 就是 Servlet。当 JSP 被 Web 容器加载时, JSP 首先会被解析成为 Servlet。因此对于 JSP 中使用的<jsp:useBean>标签也会被解析成为 Servlet 中相应的 Java 代码。

下面是在 JSP 中使用了<jsp:useBean>标签的一段代码。

```
<jsp:useBean id="mybean" class="com.MyBean" scope="session">
```

以上代码会按照如下步骤被解析为 Servlet:

(1) 首先声明一个由<jsp:useBean>中 id 属性所指定名称的句柄, 即为 mybean。该句柄的类型由<jsp:useBean>中 class 属性所指定, 即为 com.MyBean, 如下所示。

```
com.MyBean mybean = null;
```

(2) 由于指定的 scope 为 session, 所以将从相应的 Session 对象中获取对应的 JavaBean 对象, 并用 mybean 句柄指向该对象, 相应代码如下所示。

```
HttpSession scope=request.getSession();//request 为请求对象句柄, doGet/doPost 方法参数  
mybean = ( com.MyBean) scope.getAttribute("mybean ");
```

(3) 如果此时 mybean 为空, 则创建 com.MyBean 对象, 并使用 Session 对象的 setAttribute 方法将该对象存放进 Session, 如下所示。

```
if (mybean==null) {  
    mybean=new com.MyBean();  
    scope.setAttribute("mybean", mybean);  
}
```

如果 scope 的属性为其他值: request 或 application, 则只要修改步骤“2”中的第一行代码, 这两种 scope 对应的代码分别为:

- request

```
HttpServletRequest scope= request; //request 为请求对象句柄, doGet/doPost 方法参数
```

- application

```
ServletContext scope =this.getServletContext();//this 指 Servlet 对象自身句柄
```

提示 若 scope 为 page, 则保存的 JavaBean 只在 servlet 内有效, 直接声明使用即可, 不需要 setAttribute/getAttribute。

通过以上分析可以看出, 对于在 JSP 中仅仅使用一个<jsp:useBean>标签就可以完成的功能, 在 Servlet 中却要如此麻烦。因此在 JSP 中使用 JavaBean, 确实比在 Servlet 中使用 JavaBean 要方便的多。

如果要在 Servlet 中使用 JavaBean, 可以使用上面的方法, 但这样做会很麻烦。一般的解决方案是, 在 JSP 中使用<jsp:useBean>创建 JavaBean 对象并存储在 HttpSession (也可以是 HttpServletRequest 或 ServletContext) 对象中。当要在 Servlet 获取该 JavaBean 中的数据时,

只需从对应的对象中获取该 JavaBean 对象即可。

17.6.2 开发 JavaBean

本节将开发此例中用到的 JavaBean，按照如下步骤完成 JavaBean 的开发。

- (1) 启动 NetBeans，打开前面创建的项目 DemoJavaBeanApplctin。
- (2) 向 com.netbeans.web 包中添加一个名称为 MessageBean 的“Java Class”。
- (3) 按照表 17-5 所示向 MessageBean 中添加一些成员变量。

表 17-5 向 MessageBean 中添加成员变量

变 量 名 称	类 型	访问限制修饰符	初 始 值
nickName	String	private	null
title	String	private	null
OICQ	String	private	null
content	String	private	null

- (4) 为上面添加的成员变量设置相应的 get 与 set 方法。
- (5) 保存并编译该文件，此时就已经完成了 JavaBean 的开发。

17.6.3 开发 Servlet

本节将开发此例中用到的 Servlet，按照如下步骤完成 Servlet 的开发。

- (1) 向 com.netbeans.web 包中添加一个名称为 UseMessageBean，URL 映射值为 /UseMessageBean 的 Servlet。
- (2) 将下列代码添加到 processRequest 方法的方法体内（本段代码从 HttpSession 中获取 MessageBean 对象，并输出其中的内容）。

```
HttpSession session=request.getSession();
//从 session 中获取 JavaBean 对象
MessageBean messagebean=(MessageBean)session.getAttribute("message");
out.println("<h4>在 Servlet 中查看 JavaBean 中的内容:<br>");
out.println("用户昵称: "+messagebean.getNickName()+"<br>");
out.println("留言标题: "+messagebean.getTitle()+"<br>");
out.println("OICQ 号码: "+messagebean.getOICQ()+"<br>");
out.println("留言内容<br>");
out.println("<textarea cols=30 rows=5 name=backMessages>");
out.println(messagebean.getContent());
out.println("</textarea><br>");
out.println("<a href=showmessage.jsp>在 JSP 中查看 JavaBean 中数据</a>");
out.close();
```

完成了上述步骤，就已经成功开发了一个使用 JavaBean 的 Servlet。

17.6.4 开发 HTML 与 JSP

本节将开发此例中用到的 HTML 与 JSP 页面，按照如下步骤完成 HTML 与 JSP 的开发。

- (1) 向 DemoJavaBeanApplctin 中添加一个名称为 createmessage 的 HTML 文件。

(2) 将下列代码添加到<body></body>标记内。

```
<form action="showmessage.jsp" name="myform" method="post">
  用户昵称: <input type="text" name="nickName"><br>
  留言主题: <input type="text" name="title"><br>
  OICQ 号码: <input type="text" name="OICQ"><br>
  留言内容: <br><textarea cols="30" rows="5" name="content"></textarea><br>
  <input type="submit" value="提交">
</form>
```

(3) 向 DemoJavaBeanApplictin 中添加一个名称为 showmessage 的 JSP。

(4) 在编辑器中打开 showmessage.jsp 文件, 删除其中自动生成的代码, 并将下列代码添加到其中。

```
<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    <title>在 JSP 中显示 JavaBean 中数据</title>
  </head>
  <body>
    <h4>在 JSP 中查看 JavaBean 中的内容<br>
    <%
      request.setCharacterEncoding("gb2312");
    %>
    <jsp:useBean id="message" class="com.netbeans.web.MessageBean" scope="session"/>
    <jsp:setProperty name="message" property="*" />
    用户昵称: <jsp:getProperty name="message" property="nickName"/><br>
    留言主题: <jsp:getProperty name="message" property="title"/><br>
    OICQ 号码: <jsp:getProperty name="message" property="OICQ"/><br>
    留言内容: <br><textarea cols="30" rows="5" name="content">
      <jsp:getProperty name="message" property="content"/>
    </textarea><br>
    <a href="UseMessageBean">在 Servlet 中查看 JavaBean 中数据</a>
    <%//将 MessageBean 的对象 message 存储到 session 中
      session.setAttribute("message",message);
    %>
  </h4>
</body>
</html>
```

提示 href 属性的值 UseMessageBean 是该链接要转到的 Servlet 的 URL 映射值。

(5) 将 createmessage.html 设置为 DemoJavaBeanApplictin 的欢迎页面。

(6) 运行此项目, Netbeans 会自动启动 Tomcat (如果此时 Tomcat 没有启动), 并在浏览器中打开 createmessage.html 页面。输入一定内容后, 浏览器中内容如图 17-15 所示。

(7) 单击“提交”按钮，浏览器中内容如图 17-16 所示。

(8) 单击“在 Servlet 中查看 JavaBean 中数据”超链接，浏览器中内容如图 17-17 所示。



图 17-15 浏览器中内容

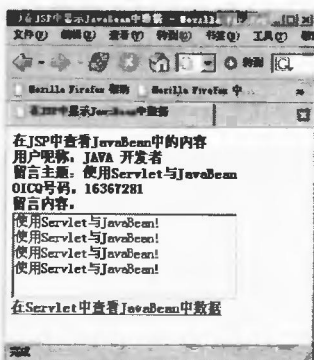


图 17-16 在 JSP 中查看 JavaBean 数据

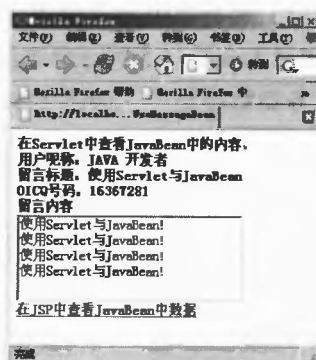


图 17-17 在 Servlet 中查看 JavaBean 数据

本例在 Servlet 中使用 HttpSession 获取 JavaBean 对象，然后输出 JavaBean 对象中的数据。还可以使用 Servlet 将从客户端接收的数据存储到 HttpSession 中，然后在其他的 Servlet（或 JSP）中获取这些数据。

17.7 HTTP 监视器

NetBeans 提供了内置的 HTTP 监视器来监视 Web 服务中 JSP/Servlet 之间传递参数的情况，不需要向 Web 应用中另外添加逻辑去跟踪 HTTP 请求以及相关状态信息，大大方便了 Web 应用的开发与调试。

如果已经在 NetBeans 中配置了 Web 服务器（如 Tomcat、Weblogic 以及 JBoss 等），HTTP 监视器会自动记录所有向 Web 服务器发送的 HTTP 请求。对于每个 HTTP 请求，HTTP 监视器同时记录请求和 Web 服务器中维护的状态信息。

使用 HTTP 监视器可以对 HTTP 请求进行分析，并且可以将 GET 与 POST 请求保存用于未来的会话分析。开发人员可以对这些请求进行编辑与再现。

17.7.1 设置 HTTP 监视器

在 NetBeans 中运行 Web 应用程序时，HTTP 监视器一般是自动启动的，它会在 NetBeans 的界面下方，如图 17-18 所示。

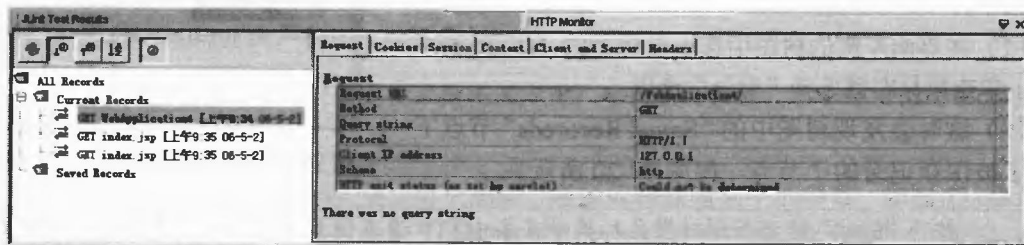


图 17-18 HTTP 监视器界面

在 HTTP 监视器界面的左侧列出了所有被监视的请求，选中其中的一个（如：GET index.jsp），

则在右侧会显示请求的细节内容。如果运行 Web 应用程序时 HTTP 监视器没有打开,则可以按照如下步骤启动。

(1) 展开“Runtime”窗口的“Servers”节点,选择要启动 HTTP 监视器的服务器(如: Bundled Tomcat 5.5.9),单击鼠标右键,在弹出的菜单中选择“Properties”选项,系统会弹出服务器管理窗口,如图 17-19 所示。

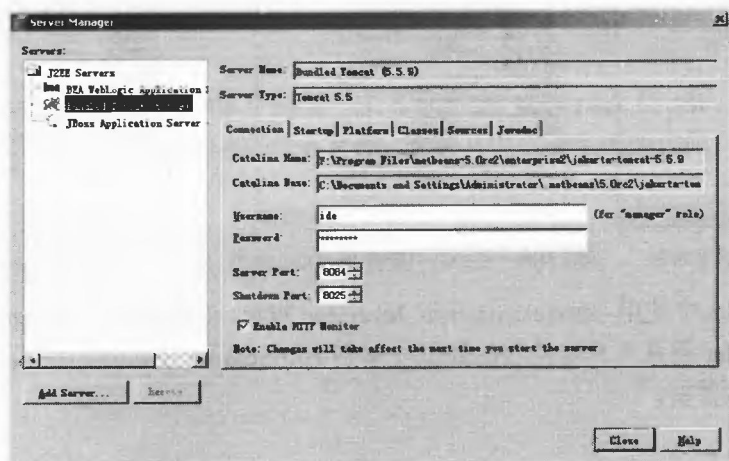


图 17-19 服务器管理窗口

(2) 在窗口的“Connection”页中选择“Enable HTTP Monitor”选项。

(3) 重新启动相应的 Web 服务器,HTTP 监视器会自动打开。

17.7.2 分析 HTTP 请求

启动了 HTTP 监视器后,在运行 Web 应用程序时,HTTP 监视器会详细记录每个请求中的数据,HTTP 监视器界面包括两部分。左侧是 HTTP 请求的树图。在树图中,有两种可查看的记录类型:当前记录(Current Records)和已保存记录(Saved Records)。当前记录类别中显示从 NetBeans 启动之后所记录的 HTTP 请求。当前记录类别中的记录在重启 Web 服务器过程中会保留,但在重启 NetBeans 的过程中不会保留。

如果希望在重新启动 NetBeans 后 HTTP 请求记录可以保留,需要对相应的 HTTP 请求记录进行保存操作,步骤如下。

(1) 在界面左侧的树图中选中要保留的 HTTP 请求记录,单击鼠标右键选择“Save”选项。

(2) 这时会发现树图中的“Saved Records”节点下出现了被保存记录的子节点,如图 17-20 所示。

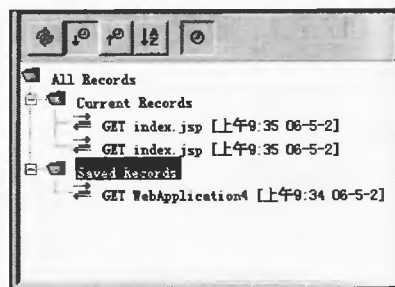


图 17-20 保存后的 HTTP 请求记录








提示 按下“Shift”键可以同时用鼠标选中多条 HTTP 请求记录进行保存。选中相应的记录,选择右键菜单中的“Delete”选项可以删除选中的记录。

(3) 在树图的上方有一系列按钮,可以对记录进行相应的操作,详细说明如表 17-6 所列。

表 17-6

各个按钮的功能

按钮	功 能	说 明
	重新加载	重新载入当前保留的所有 HTTP 请求记录
	降序排列	按照 HTTP 请求记录的时间戳降序排列
	升序排列	按照 HTTP 请求记录的时间戳升序排列
	字母顺序排列	按照字母顺序排列 HTTP 请求记录
	显示/隐藏时间戳	显示或隐藏 HTTP 请求记录的时间戳

HTTP 监视器右侧的面板中显示的是左册树图中选中的 HTTP 请求记录的详细信息，采用了 tab 页面进行分类。这些详细信息包括请求的详细信息、Cookie 的名称和值、会话数据、Servlet 上下文环境、上下文属性、客户/服务器信息（如客户端协议、客户端 IP 地址等）、以及 HTTP 请求头信息。图 17-21 显示了某条选中记录的详细信息。

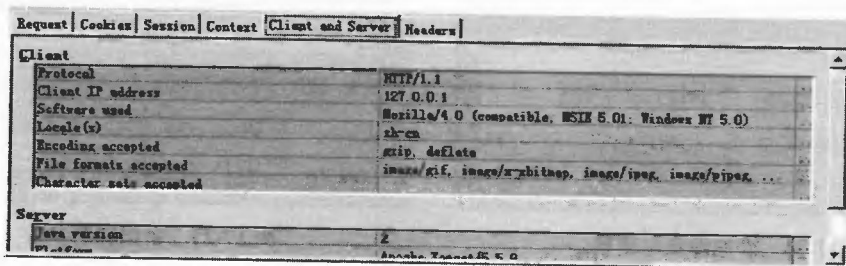


图 17-21 记录的详细信息

17.7.3 重现 HTTP 请求

HTTP 监视器最强大的功能就是编辑与重现 HTTP 请求，这样可以大大提高调试的速度，方便开发。在重现 HTTP 请求的时候，响应会出现在 Web 浏览器中。

提示 重现 HTTP 请求时，NetBeans 会自动打开默认的 Web 浏览器来显示结果。

当前记录与已保存记录都可以重现。重现一个 HTTP 请求的方法如下，在左侧树图中选择要重现的 HTTP 请求记录，选择右键菜单中的“Replay”选项即可。开发人员还可以在重现 HTTP 请求之前对其进行编辑，方法如下。

(1) 在左侧树图中选中一个 HTTP 请求，选择右键菜单的“Edit and Replay...”选项，系统会弹出编辑并重现窗口，如图 17-22 所示。

提示 在弹出的窗口中可以修改请求的不同方面，包括查询参数、请求的 URL、Cookie 内容等。

(2) 在窗口中选中“Query”页，可以编辑、增加或删除请求中的参数，如图 17-23 所示。

(3) 单击“Add Parameter”按钮，系统将弹出添加参数对话框，如图 17-24 所示。

(4) 在对话框中填写参数的名称 (Name) 和值 (Value)，单击“OK”按钮，完成添加。这时会发现，编辑与重现窗口中的“Query”页中出现了新添加的参数，如图 17-25 所示。

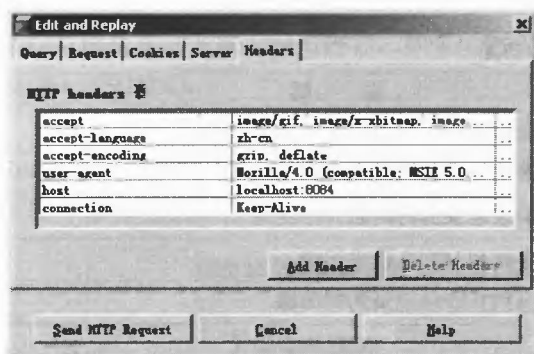


图 17-22 编辑并重现实窗口

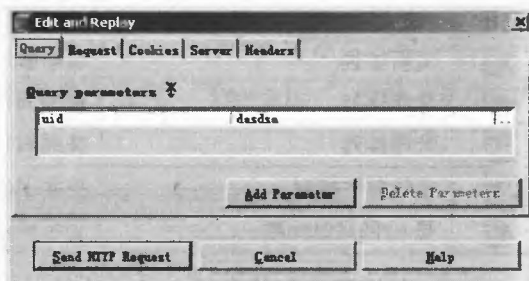


图 17-23 编辑请求参数

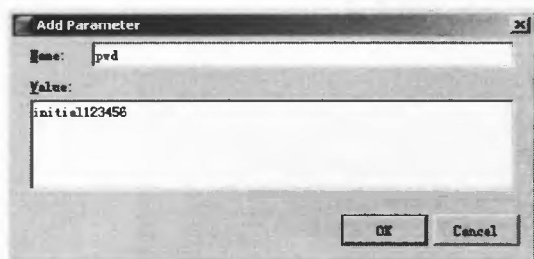


图 17-24 添加参数对话框

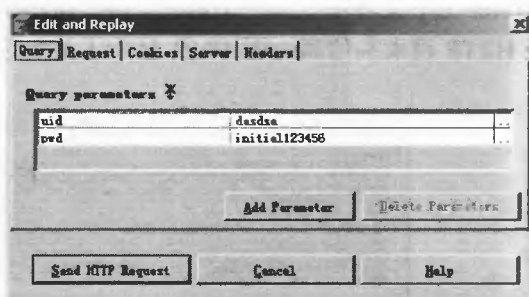


图 17-25 新添加的参数

- (5) 在参数列表中双击某个表格单元还可以对其内容进行编辑，读者可以自行操作。
- (6) 选择编辑与重现窗口中的不同页面，对其他请求信息进行编辑。
- (7) 完成对请求信息的编辑后，单击编辑与重现窗口中的“Send HTTP Request”按钮，即可进行请求的重现。重现的结果将自动显示在默认的浏览器中。

17.8 小结

本章首先对 JavaBean 组件模型的概念进行了简单的介绍，然后对其几个属性进行了简单的说明，包括索引属性、简单属性、绑定属性以及约束属性等，另外还介绍了如何在 JSP 中使用 JavaBean、在 JavaBean 中封装数据库逻辑、使用 JavaBean 实现购物车等内容。在下一章中，将开发一个综合使用 JSP、Servlet 和 JavaBean 的项目。

第 18 章

开发 MVC 架构的网上商店

前面几章对开发 Web 应用程序时常用的几种技术：JSP、Servlet 及 JavaBean 进行了详细的介绍。本章将综合使用这些技术，使用 JSP+Servlet+JavaBean 开发一个基于 MVC 架构的网上商店，主要包括如下内容。

- MVC 架构介绍；
- 项目功能演示；
- 模块功能及相互关系说明；
- 查询购买模块的开发；
- 开发管理模块。

18.1 MVC 架构介绍

MVC 是开发 Web 应用程序时经常使用的一种架构，本节将对 MVC 架构进行简单的介绍，主要包括如下内容：

- JSP 的 Model 1 与 Model 2 架构；
- MVC 介绍；
- MVC 的模型；
- MVC 的视图；
- MVC 的控制器。

18.1.1 JSP 的 Model 1 与 Model 2 架构

早期的 JSP 构建 Web 应用程序有两种方法，即 JSP Model 1 和 JSP Model 2 架构。虽然这两种技术在 JSP 规范中已经不再使用，但其却广泛地应用于 Web 开发层开发群体中。

这两种 JSP 架构在某些关键的地方有所不同，主要的差异在于处理请求的方式以及由哪个组件来处理请求。对于 Model 1 架构而言，JSP 页面会完成请求的所有处理事项，负责向客户端显示输出，如图 18-1 所示。

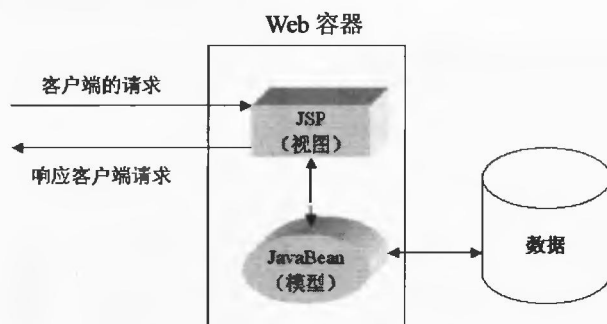


图 18-1 JSP Model 1 架构

在图 18-1 中可以看到，在 Model 1 架构中，整个流程并没有 Servlet 的参与。客户端请求直接发送到 JSP 页面，JSP 页面会再和 JavaBeans 组件或者其他服务组件通信，最后由 JSP 页面选定下一个页面。在该架构中，下一个视图由 JSP 确定，或根据客户端请求的参数确定。

相对而言，在 Model 2 架构中，客户请求开始是由一个 Servlet 负责接收，通常称此 Servlet 为控制器 Servlet。该 Servlet 会对请求做初步的处理，并决定下一步要显示的视图，如图 18-2 所示。

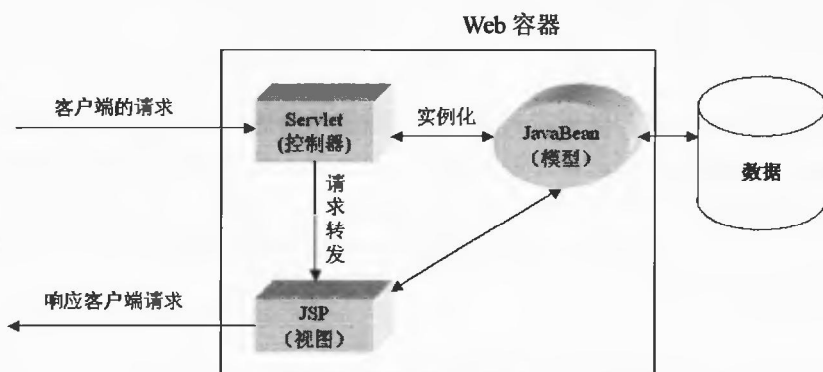


图 18-2 JSP Model 2 架构

在 Model 2 架构中，客户并不是直接把请求发送至 JSP 页面。这样可以让 Servlet 进行一些处理操作，包括认证、授权、日志管理以及方便的实现国际化。

一旦请求处理完毕，Servlet 会把请求转发到适当的 JSP 页面，至于下一个页面怎么确定，则因应用程序而异。例如，对于较简单的应用程序而言，下一页要显示的内容可能会直接硬编码在 Servlet 中；而在另一些复杂的应用中，则可能会有其他的解决方法。

可以看出，这两种架构的主要差异在于 Model 2 架构引入了一个控制器 Servlet，其提供了单一的入口点并鼓励更多的重用和可扩展性，这些都是 Model 1 的做法所不及的。采用 Model 2 架构时，表示输出和请求处理之间都有很清楚的界线。这种分离通常称为 MVC (Model-View-Controller, 模型-视图-控制器) 模式。

18.1.2 MVC 模式简介

单独就 MVC 模式来讲，其与 Web 应用程序或者 Java 并没有直接的关系。如前一节所述，

Model 2 架构和区分 Web 应用程序的责任归属有关。如果让 JSP 页面负责接收请求, 执行某个业务逻辑, 并确定下一个将要显示的页面, 会让 JSP 页面的维护和扩展都很困难。如果 Web 应用程序的不同组件各自有其明确的职责, 则应用程序的开发和维护都会变得更加容易。

MVC 在很多软件设计书籍中都归类为一种设计模式。虽然对 MVC 模式的准确定义有很多不同的看法, 但其基本定义是相同的。MVC 模式有以下 3 个主要组件。

- 模型 (model): 负责业务逻辑的状态及管理数据。
- 视图 (view): 负责显示业务领域的表示视图。
- 控制器 (controller): 负责控制用户输入的流程和状态。

使用 MVC 模式时, 通常会有某种形式的事件通知, 当模型的某部分有了变动时, 可以通知视图控件。不过由于典型的 Web 应用程序和浏览器之间的连接是无状态的, 模型对视图组件的通知很难办到。

1. MVC 模型

根据应用程序框架类型的不同, MVC 的模型部分可以有很多不同的形式。在两层应用程序中, Web 层会直接和数据存储 (如数据库) 交互, 此时模型可能是一些很平常的 Java 对象。这些对象可能会因为用户做了数据库的查询, 通过返回的 `ResultSet` 对象确定相关属性的值。在更复杂的企业级应用中, 比如 Web 层和 EJB 服务器通信, MVC 的模型部分将会是 EJB 组件。

2. MVC 视图

Web 层 MVC 的视图通常是由 HTML 和 JSP 页面组成的。HTML 页面用来提供静态内容, 而 JSP 页面则可以用来提供静态和动态内容。多数动态内容都是在 Web 层产生的。在某些应用程序中可能需要在客户端执行 JavaScript, 但这并不和 MVC 冲突。

HTML 和 JSP 并非是视图的惟一选择。例如以 WML 来替代 HTML 也很简单。因为视图和模型两者是解耦合的, 可以针对不同的客户类型支持多种视图, 但使用相同的模型组件。

3. MVC 控制器

Web 层 MVC 的控制器通常是 Java Servlet。在 Web 应用程序中, 控制器有以下职责。

- 接收客户端的 HTTP 请求。
- 把每个请求翻译成要完成的特定的业务操作。
- 自行调用业务操作, 或委托给某个处理程序。
- 确定要显示的下一个页面。
- 向请求者返回视图。

前端控制器模式是 J2EE 的一部分, 其描述了 Web 层控制器的实现方法。由于客户的所有请求和响应都会经过控制器, 因此 Web 应用程序中就会有一个集中的控制点。这样新增功能时会比较方便。

通过 MVC 模式, 以往通常要放在每个 JSP 中的程序代码, 现在可以集中放在控制器 Servlet 中, 由其处理所有的请求。控制器还会帮助把视图和业务逻辑解耦合, 使得开发更加简单。

18.2 项目功能演示

在本节以及后面的几节中，将逐步为读者讲述如何在 NetBeans 中开发一个具有 MVC 架构的网上商店。本项目包含查询购买模块与管理模块两个主要的模块，本节将对这两个模块的功能分别进行演示，主要包括如下内容：

- 查询功能演示；
- 购买功能演示；
- 管理功能演示。


18.2.1 查询功能演示

本小节将演示该项目的查询功能，操作步骤如下。

(1) 运行该项目，出现如图 18-3 所示的界面。

(2) 图 18-3 所示页面以分页的方式列出了当前的所有商品，单击“下一页”链接可以查看其他页的商品（第一页只有“下一页”链接，最后一页只有“上一页”链接，其他页同时拥有这两种链接）。

(3) 在图 18-3 所示的“查询商品名称”文本框中输入要查询的商品名称后，单击“查询”按钮进行查询。

 **说明** 查询包括精确查询与模糊查询两种状态。精确查询会从数据库中查找商品名称与用户输入的查询关键字完全匹配的商品，而模糊查询会从数据库中查找商品名称中包含用户输入的查询关键字的商品。

(4) 在“查询商品名称”文本框中输入“java”，查询方式选择“精确查询”，单击“查询”按钮，此时浏览器中内容如图 18-4 所示。

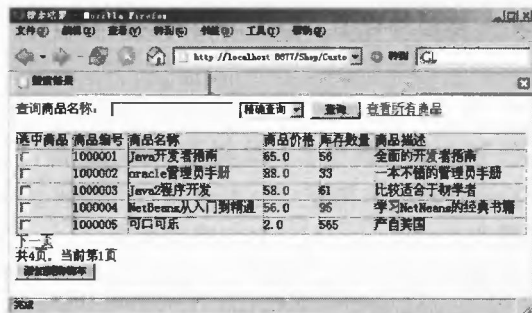


图 18-3 项目主界面

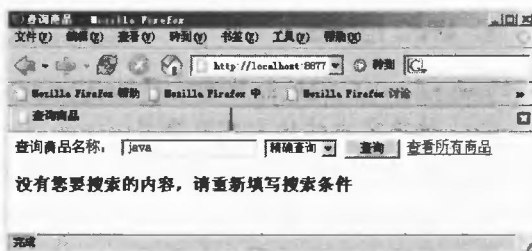


图 18-4 精确查询

(5) 可以看到在输入“java”查询关键字并选择了精确查询以后，页面会显示“没有您要搜索的内容，请重新填写搜索条件”，表明数据库中没有名称为“java”的商品。

(6) 仍然在“查询商品名称”文本框中输入“java”，但将查询方式设置为模糊查询，单击“查询”按钮后，浏览器中内容如图 18-5 所示。

(7) 从图 18-4 与 18-5 中可以看到，虽然输入了相同的查询关键字，但选择不同的查询方式很可能出现不同的查询结果。

(8) 如果输入的查询商品名称为空, 则浏览器中内容如图 18-6 所示。

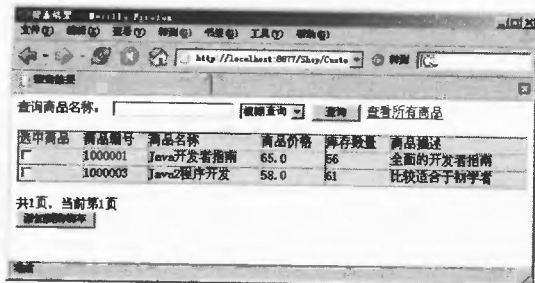


图 18-5 模糊查询

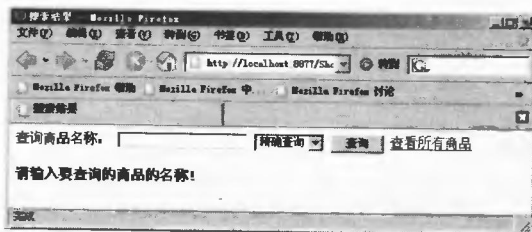


图 18-6 查询商品名称为空界面

(9) 另外, 单击“查看所有商品”链接, 此时会以分页的方式显示所有的商品, 界面与如图 18-3 所示类似。

18.2.2 购买功能演示

本小节将演示该项目的购买功能, 操作步骤如下。

(1) 在如图 18-3 所示的窗口中选中商品编号为 100002 和 100005 的商品前面的复选框, 单击“添加到购物车”按钮, 此时窗口如图 18-7 所示。

(2) 从图中可以看到, 单击“添加到购物车”按钮后, 被选中的商品会被添加到购物车中并显示出来, 并且还会显示该商品的购买数量以及商品单价。此时如果再次将编号为 100002 的商品添加到购物车中, 浏览器中内容如图 18-8 所示。



图 18-7 将商品添加到购物车

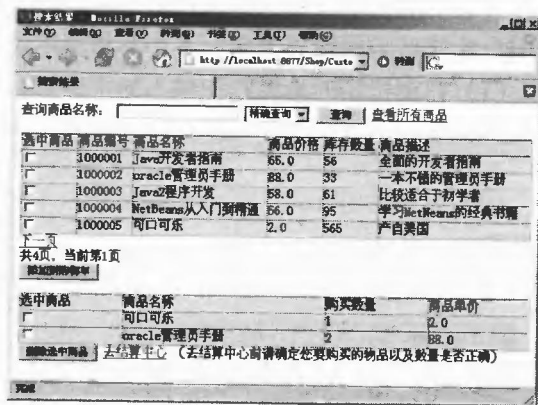


图 18-8 向购物车中添加已经存在的商品

(3) 从图 18-8 中可以看到, 如果向购物车中添加购物车中已有的商品, 则该商品的“购买数量”会被增加 (每次增加 1)。

(4) 在购物车中选中“可口可乐”前面的复选框, 单击“删除选中商品”按钮, 此时浏览器中内容如图 18-9 所示。

(5) 从图 18-9 中可以看到, 单击“删除选中商品”按钮后, 被选中的商品将会从购物车中删除。继续向购物车中添加一些商品, 然后单击“去结算中心”超级链接, 此时打开如图 18-10 所示的界面。

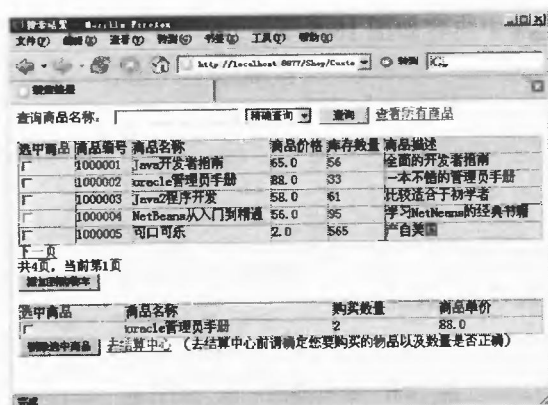


图 18-9 删除购物车中商品

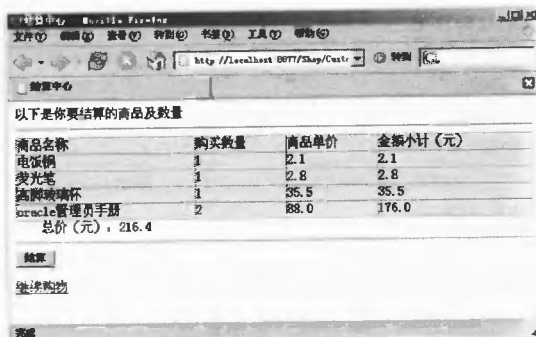


图 18-10 结算中心

(6) 图 18-10 所示的窗口给出了将要结算商品的数量、单价、金额小计以及总价。如果在如图 18-10 所示的窗口中单击“继续购物”链接，则会转到图 18-3 所示页面。如果单击“结算”按钮，则浏览器中内容如图 18-11 所示。

(7) 从图 18-11 中可以看到，结算以后，界面会显示“已经成功结算”的提示信息。此时单击“继续购物”链接，则转到图 18-3 所示页面。

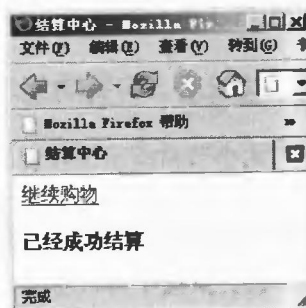


图 18-11 成功结算页面

18.2.3 管理功能演示

管理功能模块与查询购买模块不同，任何人都可以访问查询页面，而只有通过登录验证的用户才能够访问管理页面。本小节将演示该项目的管理功能，步骤如下。

(1) 在浏览器的地址栏中输入如下内容，出现登录界面，如图 18-12 所示。

`http://localhost:8877/Shop/login.jsp`

(2) 如果在图 18-12 所示页面中输入了错误的账号或密码，则会出现提示信息要求用户重新输入，如图 18-13 所示。

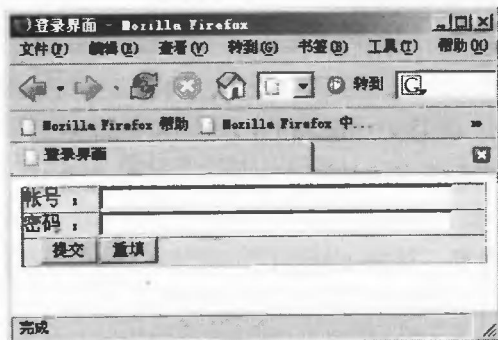


图 18-12 管理登录页面

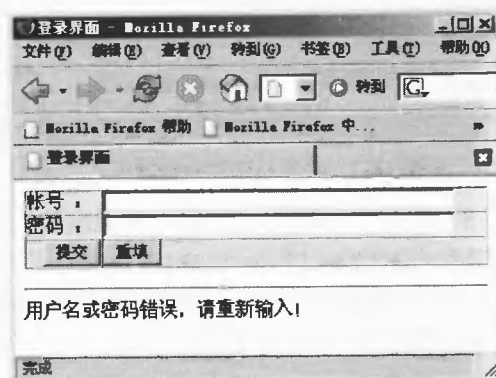


图 18-13 错误提示页面

(3) 输入正确的账号和密码以后，程序会自动跳转到查看与删除页面，浏览器中内容如

图 18-14 所示。

(4) 在图 18-14 中, 查看与删除页面显示了管理员的名称, 并以分页的方式显示了当前数据库中的所有商品。

(5) 单击“删除”链接删除相应的商品。单击编号为“100005”的商品所对应的“删除”链接后, 浏览器中内容如图 18-15 所示。

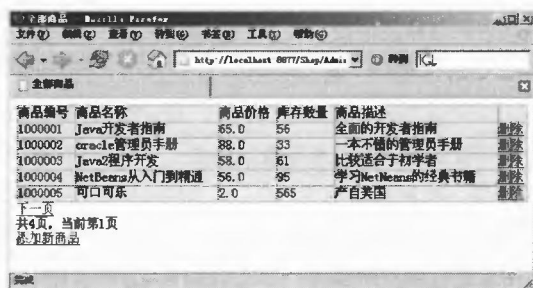


图 18-14 管理员登录后页面

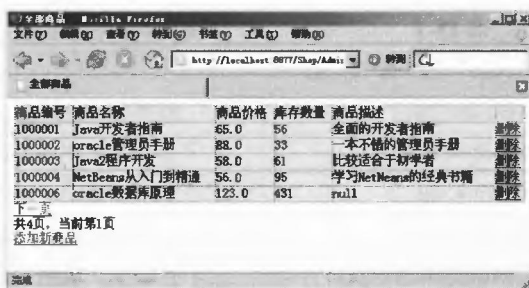


图 18-15 删除编号为 100005 的商品

(6) 从图中可以看到, 单击“删除”链接后, 编号为 100005 的商品被删除了。单击“添加新商品”链接, 浏览器中内容如图 18-16 所示。

(7) 可以在图 18-16 所示页面中设置要添加的商品的信息, 设置完成后, 单击“提交”按钮提交所添加的信息。

(8) 添加一个编号为 1000005, 名称为“可口可乐”, 商品价格为 2.0, 商品数量为 265, 商品描述为“产自美国”的商品, 单击“提交”按钮提交数据。此时单击“查看已有商品”链接, 浏览器中内容如图 18-17 所示。

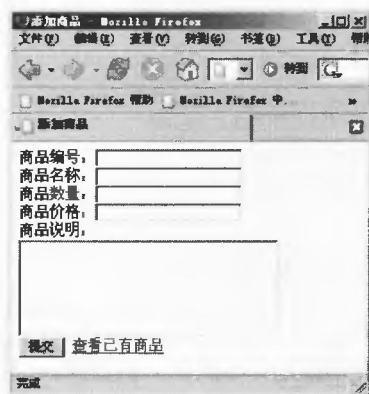


图 18-16 添加商品页面

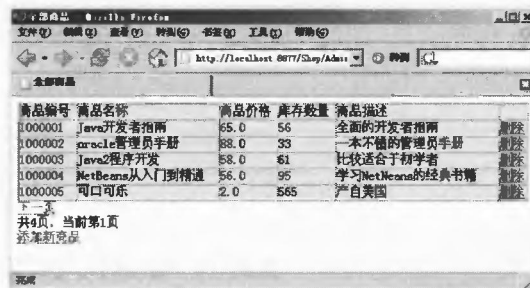


图 18-17 成功添加编号为 100005 的商品

从图 18-17 中可以看到, 编号为 1000005 的商品确实被添加到了数据库中。

18.3 模块功能及相互关系说明

上一节中已经提到, 本项目主要包括: 查询购买模块与管理模块。本节给出了相应模块各个子模块及其之间的关系, 主要包括如下内容:

- 查询购买模块;

● 管理模块。

18.3.1 查询购买模块

查询购买模块完成了用户查询商品并购买的功能,表 18-1 给出了查询购买模块的各个子模块的名称以及功能说明。

表 18-1 查询购买模块的各个子模块的名称及说明

文件名称	说 明
welcome.jsp	程序的欢迎界面
searchview.jsp	该页面用来查询商品,并以分页的方式显示查找结果
cartview.jsp	显示购物车中的商品
checkout.jsp	显示用户选定的商品以及数量,并在此页进行结算
CustomerServlet.java	控制器,根据用户的请求进行相应的操作
ProductBean.java	表示商品信息的 JavaBean
DataBaseBean.java	封装了与数据库有关的操作的 JavaBean
CartBean.java	封装购物车信息的 JavaBean
CartProduct.java	表示购物车中的商品信息的 JavaBean

图 18-18 给出了查询购买模块中各个子模块之间的关系。

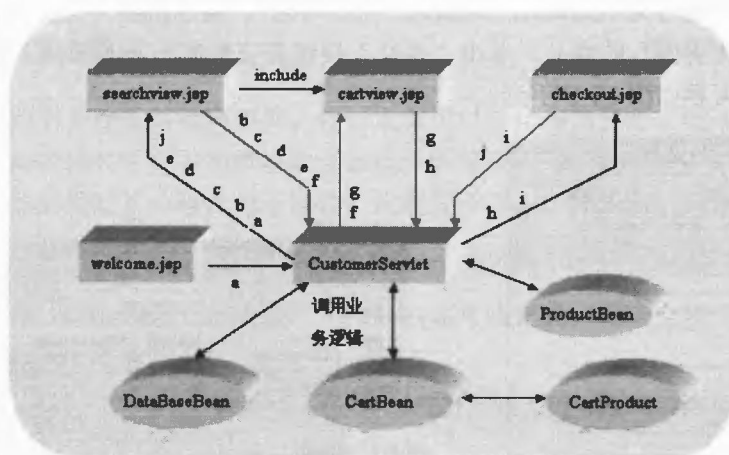


图 18-18 查询购买模块各个子模块间的关系

表 18-2 给出了各个编号所对应的动作内容。

表 18-2 动作编号对照表

动作编号	动作内容	动作编号	动作内容
a	显示所有商品	b	查询商品
c	查看上一页商品	d	查看下一页商品
e	查看所有商品	f	添加到购物车
g	删除选中商品	h	去结算中心
i	结算商品	j	继续购物

18.3.2 管理模块

管理模块用来完成添加与删除商品,表 18-3 给出了管理模块中相应子模块的名称以及功能说明。

表 18-3 管理模块的各个子模块的名称及说明

文件名称	说明
login.jsp	管理员登录验证页面,如果验证不成功还回到该页面
showadddelete.jsp	以分页的方式显示当前所有的商品,还可以在该页面中删除商品
addproduct.jsp	添加商品页面
AdminServlet.java	控制器,根据客户端的请求进行相应的操作
AdminDataBaseBean.java	封装了与数据库有关的操作的 JavaBean 组件

另外,管理模块中还用到了 ProductBean,其功能已经在查询购买模块中给出,这里不再赘述。

图 18-19 给出了管理模块中各个子模块之间的关系。

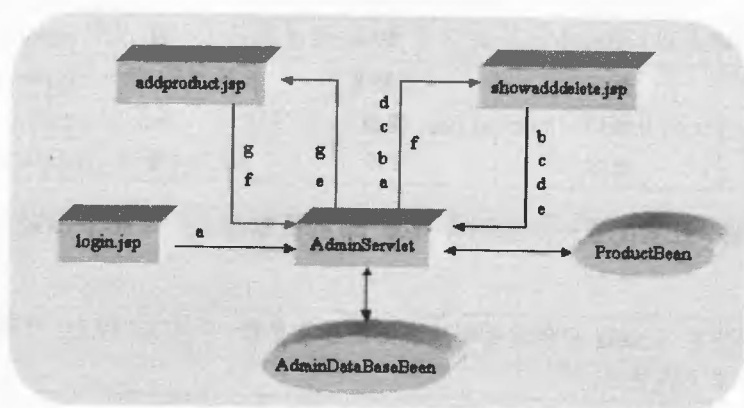


图 18-19 管理模块子模块的关系

表 18-4 给出了各个编号所对应的动作内容。

表 18-4 动作编号对照表

动作编号	动作内容	动作编号	动作内容
a	查看已有商品	b	删除商品
c	获取上一页的商品	d	获取下一页的商品
e	转到添加商品页	f	查看已有商品
g	添加商品		

18.4 查询购买模块的开发

本节将带领读者完成查询购买模块的开发,主要包括如下内容:

- 数据库准备工作;

- 开发 ProductBean、CartProduct 与 CartBean;
- 开发 DataBaseBean;
- 开发查询购买模块的 Servlet;
- 开发 searchview.jsp;
- 开发 cartview.jsp 与 checkout.jsp;
- 开发 welcome.jsp。

18.4.1 数据库准备工作

查询购买模块中使用了数据库,因此在开发项目之前,应进行与数据库有关的准备工作,即创建数据库表与配置数据源。

本模块中,用 Access 创建了一个名称为 shop 的数据库,并向数据库中添加了一个名称为 product 的表。表 product 各个字段名称及其属性如表 18-5 所示。

表 18-5 product 表的各个字段及其属性值

字段名称	数据类型	字段大小	允许空字符串	必填字段
product_id	数字	长整型		是
product_name	文本	50	否	是
product_price	数字	双精度		是
product_num	数字	整型		是
product_describe	文本	255	是	否

创建完表以后,还要配置一个名称为 shop_db 的数据源,并将 shop 指定为该数据源所指向的数据库。

提示 关于如何在 Access 中创建数据库表与配置数据源,已经在 17.4.2 节进行了详细的介绍,这里不再赘述。

18.4.2 开发 ProductBean、CartProduct 与 CartBean

查询购买模块中包含 ProductBean、CartProduct、CartBean 和 DataBaseBean 4 个 JavaBean 组件。本小节将完成前 3 个 JavaBean 组件的开发,对于 DataBaseBean 的开发过程,将在下个小节中介绍。按照如下步骤完成这 3 个 JavaBean 组件的开发。

(1) 在 NetBeans 中创建一个名称为 Shop 的 Web 项目,并向其中添加一个名称为 org.netbeans.Web 的包。

(2) 向 org.netbeans.Web 包中添加一个名称为 ProductBean 的 Java Class。

(3) 按如表 18-6 所示向 ProductBean 中添加成员变量。

表 18-6 ProductBean 中成员变量的各选项说明

变量名称	变量类型	访问限制修饰符号	初始值
productId	int	private	null
productName	String	private	null
productPrice	double	private	null

续表

变量名称	变量类型	访问限制修饰符号	初始值
productNum	int	private	null
productDescribe	String	private	null

(4) 为步骤(3)中所添加的成员变量设置相应的 get 与 set 方法。

(5) 向 org.netbeans.Web 中添加一个名称为 CartProduct 的 Java Class。

(6) 按如表 18-7 所示向 CartProduct 中添加成员变量。

表 18-7 CartProduct 中成员变量的各选项说明

变量名称	变量类型	访问限制修饰符号	初始值
productId	int	private	null
productName	String	private	null
selectedCount	int	private	null
productPrice	double	private	null

(7) 为步骤(6)中添加的成员变量设置 get 与 set 方法。

(8) 向 org.netbeans.Web 中添加一个名称为 CartBean 的 Java Class。

(9) 向 CartBean 中添加一个名称为 hash, 类型为 Hashtable, 访问限制修饰符号为 private, 初始值为 new Hashtable() 的成员变量。

(10) 向 CartBean 中添加一个签名为 private void productNotExit(Vector vec,String id) 的方法, 并将下列代码添加到该方法中:


```
//获取当前查询出来的所有商品,并根据其 ID 号重新组织 CartProduct 的内容
Enumeration enume=vec.elements();
int tempid=Integer.parseInt(id);
while(enume.hasMoreElements()){
    ProductBean pb=(ProductBean)enume.nextElement();
    if(tempid==(pb.getProductid())){
        CartProduct cp=new CartProduct();
        cp.setProductid(pb.getProductid());
        //从 Vector 中获取具有相同 Id 号码的商品名称,
        //作为 CartProduct 的 productName 属性的值
        cp.setProductName(pb.getProductName());
        cp.setSelectedCount(1);
        cp.setProductPrice(pb.getProductPrice());
        hash.put(id,cp);
        return ;
    }
}
```

注意

如果当前购物车中不存在指定 ID 的商品时调用该方法,即将该商品添加到购物车中。
指定 ID 的商品的名称从当前查询结果 (Vector 对象) 中获取。

(11) 向 CartBean 中添加一个签名为 public void addToCart(Vector vec,String id) 的方法, 将下列代码添加到该方法中:


```
if (hash.get(id) != null) {
    CartProduct tempcp = (CartProduct) hash.get(id);
    CartProduct cp = new CartProduct();
    cp.setProductId(tempcp.getProductId());
    cp.setProductName(tempcp.getProductName());
    cp.setSelectedCount((tempcp.getSelectedCount() + 1));
    cp.setProductPrice(tempcp.getProductPrice());
    hash.remove(id);
    hash.put(id, cp);
} else {
    this.productNotExist(vec, id);
}
```

 **说明** 该方法的作用是如果当前购物车中存在相同 ID 的商品，则将该商品的数量加 1。

(12) 向 CartBean 中添加一个签名为 `public Hashtable getCartContent()` 的方法，该方法用来获取购物车中的内容，返回值为 `Hashtable` 对象。将下列代码添加到该方法中：

```
return this.hash;
```

(13) 向 CartBean 中添加一个签名为 `public void deleteFromCart(String id)` 的方法，该方法用来删除购物车内指定 ID 的商品。并将下列代码添加到该方法中：

```
this.hash.remove(id);
```

(14) 向 CartBean 中添加一个签名为 `public void clearCart()` 的方法，该方法用来清空购物车。将下列代码添加到该方法中：

```
this.hash.clear();
```

(15) 向 CartBean 中添加一个签名为 `public boolean isEmpty()` 的方法，该方法用来判断当前购物车是否为空。将下列代码添加到该方法中：

```
boolean empty = false;
if (this.hash.isEmpty()) {
    empty = true;
} else {
    empty = false;
}
return empty;
```

完成了上述步骤，就已经成功地完成了这 3 个 `JavaBean` 组件的开发。

18.4.3 开发 DataBaseBean

`DataBaseBean` 封装了与数据库有关的操作，本小节将带领读者完成 `DataBaseBean` 的开发。按照如下步骤完成。

(1) 向 `org.netbeans.Web` 中添加一个名称为 `DataBaseBean` 的 `Java Class`。

(2) 向 `DataBaseBean` 中添加一个名称为 `con`，类型为 `Connection`，修饰限制符号为 `private`，初始值为 `null` 的成员变量。

(3) 向 DataBaseBean 中添加一个签名为 `private void connectTODB()` 的方法, 将下列代码添加到该方法中:

```
String CLASSFORNAME="sun.jdbc.odbc.JdbcOdbcDriver";
String CONNECTSTR="jdbc:odbc:shop_db";
try{
    Class.forName(CLASSFORNAME);
    this.con=DriverManager.getConnection(CONNECTSTR);
} catch(Exception e) {
    e.printStackTrace();
}
```

说明 该方法用来获取数据库连接, 并将该连接赋值与 con。

(4) 向 DataBaseBean 中添加一个签名为 `public Vector selectProduct(String sql)` 的方法, 并将下列代码添加到该方法中:

```
Statement stmt=null;
ResultSet rest=null;
Vector vec=new Vector();
this.connectTODB();
try{
    stmt=this.con.createStatement();
    rest=stmt.executeQuery(sql); //从数据库中查询指定 ID 的产品
    while(rest.next()){
        ProductBean temppro=new ProductBean();
        temppro.setProductId(rest.getInt("product_id"));
        temppro.setProductName(rest.getString("product_name"));
        temppro.setProductPrice(rest.getDouble("product_price"));
        temppro.setProductNum(rest.getInt("product_num"));
        temppro.setProductDescribe(rest.getString("product_describe"));
        //将查询出的商品组织成 ProductBean 对象后, 添加到 Vector 对象中返回
        vec.add(temppro);
    }
} catch(Exception e){
    e.printStackTrace();
} finally{//关闭数据库相关连接
    try {
        if(rest!=null){
            rest.close();
        }
        if(stmt!=null){
            stmt.close();
        }
        if(this.con!=null){
            con.close();
        }
    }
```

```

        } catch (Exception ee) {
            ee.printStackTrace();
        }
    }
    return vec;
}

```



说明

该方法用于从数据库中查询具有指定 ID 的商品。将查询得到的结果封装成 `ProductBean` 对象后添加到 `Vector` 对象中返回。

(5) 向 `DataBaseBean` 中添加一个签名为 `public void checkOut(CartBean cartbean)` 的方法, 将下列代码添加到该方法中:

```

Statement stmt=null;
Hashtable hash=cartbean.getCartContent();
Enumeration cartEnu=hash.elements();
this.connectTODB();
while(cartEnu.hasMoreElements()) {
    CartProduct cartpro=(CartProduct)cartEnu.nextElement();
    int id=cartpro.getProductId();
    int num=cartpro.getSelectedCount();
    try {
        stmt=con.createStatement();
        stmt.executeUpdate("update product set product_num=product_num-
+num+\" where product_id=\"+id+\" and product_num>"+num);
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
try {
    if(stmt!=null){
        stmt.close();
    }
    if(this.con!=null){
        con.close();
    }
} catch (Exception ee) {
    ee.printStackTrace();
}
}

```



说明

该方法用于在客户确定购买后从数据库中将指定 ID 的商品的库存数量减去给定的值。

完成了上述步骤, 就已经成功地完成了 `DataBaseBean` 的开发。

18.4.4 开发查询购买模块的 Servlet

本项目中查询购买模块的 Servlet 为 `CustomerServlet`, 按照如下步骤完成 `CustomerServlet` 的开发。

(1) 打开前面创建的项目 Shop, 向包 org.netbeans.Web 中添加一个名称为 CustomerServlet, URL 映射值为 /CustomerServlet 的 Servlet。


(2) 向 CustomerServlet 中添加一个签名为 private void forward(HttpServletRequest request, HttpServletResponse response, String url) throws ServletException, IOException 的方法, 该方法用来将请求转发到由 url 指定的页面。将下列代码添加到该方法中:

```
RequestDispatcher dispatcher = request.getRequestDispatcher(url);
dispatcher.forward(request, response);
```

(3) 删除 processRequest 方法内自动生成的代码, 并将下列代码添加到该方法中:

```
response.setContentType("text/html;charset=gb2312");
request.setCharacterEncoding("gb2312");
HttpSession session=request.getSession(true);
String action=request.getParameter("action").trim();//获取 action 属性的值
//获取用户选择的复选框的值
String [] selectedCount=request.getParameterValues("choose");
DataBaseBean dbBean=new DataBaseBean();
Vector vec=new Vector();
if(action.equals("select")){//处理用户按下“查询”按钮的请求
    String productName=request.getParameter("productName");
    String searchType=request.getParameter("searchType");
    if(productName==null|| productName.equals("")){//输入的查询关键字为空
        session.setAttribute("namenull","namenull");
        vec=dbBean.selectProduct("select * from product");
    }else if(searchType.equals("jingque")){//查询方式为“精确查询”
        vec=dbBean.selectProduct("select * from product where product_
name like '"+productName+"'");
    } else if(searchType.equals("mohu")){//查询方式为“模糊查询”
        vec=dbBean.selectProduct("select * from product where product_
name like '%" +productName+"%'");
    }
    session.setAttribute("result",vec);
    this.forward(request,response,"searchview.jsp");
}else if(action.equals("getall")){//处理用户单击“查看所有商品”链接的请求
    vec=dbBean.selectProduct("select * from product");
    session.setAttribute("result",vec);
    this.forward(request,response,"searchview.jsp");
}else if(action.equals("next")){//处理用户单击“下一页”链接的请求
    String pageno=request.getParameter("pageno");
    this.forward(request,response,"searchview.jsp?pageno="+pageno);
}else if(action.equals("previous")){//处理用户单击“上一页”链接的请求
    String pageno=request.getParameter("pageno");
    this.forward(request,response,"searchview.jsp?pageno="+pageno);
}else if(action.equals("addtocart")){//处理用户按下“添加到购物车”按钮的请求
    if(selectedCount!=null){
        CartBean cartbean=null;
        Vector tempvec=(Vector)session.getAttribute("result");
```

```
        if(session.getAttribute("cart")==null){
            cartbean=new CartBean();
        }else{
            cartbean=(CartBean)session.getAttribute("cart");
        }
        for(int i=0;i<selectedCount.length;i++){
            cartbean.addToCart(tempvec,selectedCount[i]);
        }
        session.setAttribute("cart",cartbean);
    }
    this.forward(request,response,"cartview.jsp");
}else if(action.equals("delete")){//处理用户按下“删除选中商品”按钮的请求
    if(selectedCount!=null){
        CartBean cartbean=(CartBean)session.getAttribute("cart");
        for(int i=0;i<selectedCount.length;i++){
            cartbean.deleteFromCart(selectedCount[i]);
        }
        session.setAttribute("cart",cartbean);
    }
    this.forward(request,response,"cartview.jsp");
}else if(action.equals("gotocheckout")){//处理用户单击“去结算中心”链接的请求
    this.forward(request,response,"checkout.jsp");
}else if(action.equals("checkout")){//处理用户按下“结算”按钮的请求
    CartBean cartbean=(CartBean)session.getAttribute("cart");
    dbBean.checkOut(cartbean);
    session.setAttribute("cart",null);
    session.setAttribute("check","ok");
    this.forward(request,response,"checkout.jsp");
}
```

 **说明** 以上代码根据客户端的请求进行相应的操作。

完成了上述步骤，就已经成功地完成了 CustomerServlet 的开发。

18.4.5 开发查询与显示页面

本项目中的查询与显示页面为 searchview.jsp。可以在 searchview.jsp 页面中查询商品，并以分页的方式显示查询结果。按照如下步骤完成 searchview.jsp 的开发。

(1) 打开前面创建的项目 Shop，向其中添加一个名称为 searchview 的 JSP。

(2) 在编辑器窗口中打开 searchview.jsp 文件，删除其中自动生成的代码，并将下列代码添加到其中：

```
<%@page contentType="text/html" autoFlush="true"%>
<%@page pageEncoding="gb2312"%>
<%@page import="java.sql.*"%>
<%@page import="java.util.*"%>
<%@page import="org.netbeans.Web.*"%>
```



```

<%//声明两个变量
    int pageRecordNum=5;
    int pageCount;
%>
<%
    String tempPageNum=request.getParameter("pageno");
    try{
        if(tempPageNum==null){//在未使用上一页,下一页之前,pageno 为 null
            pageCount=1;
        }else {
            pageCount=Integer.parseInt(tempPageNum);
        }
    }catch (Exception e){
        e.printStackTrace();
    }
%>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    <title>搜索结果</title>
</head>
<body>
    <form name="search" action="CustomerServlet" method="POST">
        查询商品名称:
        <input type="text" name="productName" value="" width="40" />
        <select name="searchType">
            <option value="jingque">精确查询</option>
            <option value="mohu">模糊查询</option>
        </select>
        <input type="hidden" value="select" name="action">
        <input type="submit" value=" 查询 " />
        <a href="CustomerServlet?action=getall">查看所有商品</a>
    </form>
    <form action="CustomerServlet" method="post" name="result">
        <input type="hidden" name="pageno" value="<%=pageCount%>" />
        <%
            if(session.getAttribute("namenull")!=null){//输入懂得查询字符串为空
                out.println("<h4>请输入要查询的商品的名称!</h4>");
                session.setAttribute("namenull",null);
            }else if(((Vector)session.getAttribute("result")).isEmpty()){//查询结果为空
                out.println("<h4>没有你要搜索的内容,请重新填写搜索条件</h4>");
            }else{//查询的内容不为空
                %>
                <table align="center" width="100%" bgcolor="dddddd"
                    border="1" cellpadding="0" cellspacing="0">
                <tr>

```

精通 NetBeans——Java 桌面、Web 与企业级程序开发详解

提示 action 与 href 属性的值 CustomerServlet 是该 JSP 页面中数据要提交到的 Servlet 的 URL 映射值。

18.4.6 开发购物车页面与结算页面

(2) 在编辑器窗口中打开 `cartview.jsp` 文件，删除其中自动生成的代码，并将下列代码添加到其中：

397 第 18 章 开发 MVC 架构的网上商店

```

        int size=cartHash.size();
        Enumeration cartEnu=cartHash.elements();
        while(cartEnu.hasMoreElements()) {
            CartProduct cartproduct=(CartProduct)cartEnu.nextElement();
        }
    }
    <tr>
        <td><input type="checkbox" name="choose" value="
            <%=cartproduct.getProductId()%>" /></td>
        <td><%=cartproduct.getProductname()%></td>
        <td><%=cartproduct.getSelectedCount()%></td>
        <td><%=cartproduct.getProductPrice()%></td>
    </tr>
    <% } } %>
</table><br>
<input type="hidden" value="delete" name="action">
<input type="submit" value="删除选中商品" />
<a href="CustomerServlet?action=gotocheckout">去结算中心</a>
    (去结算中心前请确定您要购买的物品以及数量是否正确)
</form>
</body>
</html>

```

(3) 向 Shop 中添加一个名称为 checkout 的 JSP。

(4) 在编辑器窗口中打开 checkout.jsp 文件, 删除其中自动生成的代码, 并将下列代码添加到其中:

```

<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<%@page import="org.netbeans.Web.*"%>
<%@page import="java.util.*"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
        <title>结算中心</title>
    </head>
    <body>
        <%
            if((CartBean)session.getAttribute("cart")==null){
                out.println("当前购物车为空");
            }
            else{ %>
                <form action="CustomerServlet" method="post">
                    <table align="left" width="100%" bgcolor="dddddd" border="1"
                        cellpadding="0" cellspacing="0">
                        <tr align="center">以下是你要结算的商品及数量</tr><tr>

```

提示 action 与 href 属性的值 CustomerServlet 是 JSP 页面中数据要提交到的 Servlet 的 URL 映射值。

完成了上述步骤,就已经成功地完成了 cartview.jsp 与 checkout.jsp 的开发。

18.4.7 开发欢迎页面

本项目中的欢迎页面为 welcome.jsp 页面,其主要用来实现请求的转发,按如下步骤完成其开发。

(1) 向 Shop 中添加一个名称为 welcome 的 JSP。

(2) 在编辑器窗口中打开 welcome.jsp 文件,删除其中自动生成的代码,并将下列代码添加到其中:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<jsp:forward page="CustomerServlet">
    <jsp:param name="action" value="getall"/>
</jsp:forward>
```

(3) 将 welcome.jsp 设置为该项目的欢迎页面。

(4) 完成了上述步骤,就成功地完成了 welcome.jsp 的开发。到此为止,与查询购买模块有关的功能组件已经全部开发完成了。编译并运行该项目。则可以看到如图 18-3 所示界面。

18.5 开发管理模块

18.5.1 创建数据库表

管理模块中用到了数据库,因此在开发管理模块之前,应创建相应的数据库表。在 18.4.1 小节中创建的名称为 shop 的数据库中新建一个名称为 admin 的表。表 admin 中各段名称及其属性如表 18-8 所示。

表 18-8 admin 表的各个字段及其属性值

字段名称	数据类型	字段大小	允许空字符串	必填字段
admin_id	文本	文本	否	是
password	文本	50	否	是

18.5.2 开发 JavaBean

管理模块中的 JavaBean 组件仅仅包括 AdminDataBaseBean,其封装了与数据库有关的操作。按照如下步骤完成 AdminDataBaseBean 的开发。


(1) 打开前面创建的项目 Shop,向 org.netbeans.Web 中添加一个名称为 AdminDataBaseBean 的 Java Class。

(2) 向 AdminDataBaseBean 中添加一个名称为 con,类型为 Connection,访问限制修饰符号为 private,初始值为 null 的成员变量。

(3) 向 AdminDataBaseBean 中添加一个签名为 private void connectTODB()的方法,将下列代码添加到该方法中:


```
String CLASSFORNAME="sun.jdbc.odbc.JdbcOdbcDriver";
```

```
String CONNECTSTR="jdbc:odbc:shop_db";
try{
    Class.forName(CLASSFORNAME);
    this.con=DriverManager.getConnection(CONNECTSTR);
} catch(Exception e) {
    e.printStackTrace();
}
```

 **说明** 该方法主要用于获取数据库链接，并将获取的链接赋值于 con。

(4) 向 AdminDataBaseBean 中添加一个签名为 private void close (Connection con, Statement stmt, ResultSet rest) 的方法，将下列代码添加到该方法中：

```
try {
    if(rest!=null){
        rest.close();
    }
    if(stmt!=null){
        stmt.close();
    }
    if(this.con!=null){
        con.close();
    }
} catch(Exception ee){
    ee.printStackTrace();
}
```

 **说明** 该方法用来关闭与数据库有关的连接。

(5) 向 AdminDataBaseBean 中添加一个签名为 public boolean adminLogin (String username, String password) 的方法，并将下列代码添加到该方法中：

```
boolean loginok=false;
Statement stmt=null;
ResultSet rest=null;
this.connectTODB();
try {
    stmt=con.createStatement();
    rest=stmt.executeQuery("select * from admin where admin_id like '"+username+"' and password like '"+password+"'");
    if(rest.next()){
        loginok=true;
    }else{
        loginok=false;
    }
} catch (SQLException ex) {
```

```
        ex.printStackTrace();
    } finally{
        this.close(con,stmt,rest);
    }
    return loginok;
}
```

- 该方法用来从表 admin 中查询指定记录。
- 如果查询成功到该记录则返回 true, 否则返回 false。

(6) 向 AdminDataBaseBean 中添加一个签名为 public Vector getAllProduct()的方法, 并将下列代码添加到该方法中:

```
Statement stmt=null;
ResultSet rest=null;
Vector vec=new Vector();
this.connectTODB();
try{
    stmt=this.con.createStatement();
    rest=stmt.executeQuery("select * from product");
    while(rest.next()){
        ProductBean temppro=new ProductBean();
        temppro.setProductId(rest.getInt("product_id"));
        temppro.setProductName(rest.getString("product_name"));
        temppro.setProductPrice(rest.getDouble("product_price"));
        temppro.setProductNum(rest.getInt("product_num"));
        temppro.setProductDescribe(rest.getString("product_describe"));
        vec.add(temppro);
    }
}catch(Exception e){
    e.printStackTrace();
}finally{
    this.close(con,stmt,rest);
}
return vec;
}
```

- 该方法用于获取表 product 中的所有记录。
- 将获得的记录封装成 ProductBean 对象后, 添加到 Vector 对象中并返回。

(7) 向 AdminDataBaseBean 中添加一个签名为 public void deleteProduct(String id)的方法, 该方法用于从数据库中删除具有指定 id 的记录。将下列代码添加到该方法中:

```
Statement stmt=null;
this.connectTODB();
int tempid=Integer.parseInt(id);
try{
    stmt=this.con.createStatement();
    stmt.execute("delete from product where product_id="+tempid);
}catch(Exception e){
    e.printStackTrace();
}
```



```
}finally{
    this.close(con, stmt, null);
}
```

(8) 向 AdminDataBaseBean 中添加一个签名为 `public int addProduct (String id, String name, String price, String num, String describe)` 的方法, 并将下列代码添加到该方法中:

```
PreparedStatement ps=null;
int line=-1;//该变量用来记录更新记录的条数, 客户端根据该值输出一定内容
int tempid=Integer.parseInt(id);
String tempname=name;
double tempprice=Double.parseDouble(price);
int tempnum=Integer.parseInt(num);
String tempdesc=describe;
this.connectTODB();
try {
    ps=con.prepareStatement("insert into product values (?, ?, ?, ?, ?)");
    ps.setInt(1,tempid);
    ps.setString(2,tempname);
    ps.setDouble(3,tempprice);
    ps.setInt(4,tempnum);
    ps.setString(5,tempdesc);
    line=ps.executeUpdate();
} catch (SQLException ex) {
    ex.printStackTrace();
}finally{
    try{
        if(ps!=null){
            ps.close();
        }if(con!=null){
            con.close();
        }
    }catch(Exception e){
        e.printStackTrace();
    }
}
return line;
```

 **说明** 该方法用于向数据库中插入记录, 并返回插入记录的数目。

完成了上述步骤, 即完成了 JavaBean 的开发。

18.5.3 开发 Servlet

管理模块中的控制器 Servlet 为 AdminServlet, 按照如下步骤完成其开发。

(1) 打开前面创建的项目 Shop, 向 org.netbeans.Web 中添加一个名称为 AdminServlet, URL 映射值为 /AdminServlet 的 Servlet。

(2) 向 AdminServlet 中添加一个签名为 `private void forward (HttpServletRequest request, HttpServletResponse response, String url) throws ServletException, IOException` 的方法, 该方法用于将请求转发到由 url 指定的页面。将下列代码添加到该方法中:

```
RequestDispatcher dispatcher = request.getRequestDispatcher(url);
dispatcher.forward(request, response);
```

(3) 清除 `processRequest` 方法体内自动生成的代码, 并将下列代码添加到该方法中:

```
response.setContentType("text/html;charset=gb2312");
request.setCharacterEncoding("gb2312");
HttpSession session=request.getSession();
Vector vec=new Vector();
AdminDataBaseBean database= new AdminDataBaseBean();
String action=request.getParameter("action").trim();//获取 action 属性的值
if(action.equals("login")){//用户单击“提交”按钮
    String username=request.getParameter("username").trim();
    String password=request.getParameter("password").trim();
    boolean login=database.adminLogin(username,password);//查看是否通过验证
    if(login){
        vec=database.getAllProduct();
        session.setAttribute("showresult",vec);
        session.setAttribute("adminname",username);
        this.forward(request,response,"showadddelete.jsp");
    }else{
        session.setAttribute("notlogin","loginfail");
        this.forward(request,response,"login.jsp");
    }
} else if(action.equals("delete")){//用户单击“删除”链接
    String id=request.getParameter("productId");
    String pageno=request.getParameter("pageno");
    database.deleteProduct(id);
    vec=database.getAllProduct();
    session.setAttribute("showresult",vec);
    this.forward(request,response,"showadddelete.jsp?pageno="+pageno);
}else if(action.equals("next")){//用户单击“下一页”链接
    String pageno=request.getParameter("pageno");
    this.forward(request,response,"showadddelete.jsp?pageno="+pageno);
}else if(action.equals("previous")){//用户单击“上一页”链接
    String pageno=request.getParameter("pageno");
    this.forward(request,response,"showadddelete.jsp?pageno="+pageno);
}else if(action.equals("add")){//用户单击“添加新商品”链接
    this.forward(request,response,"addproduct.jsp");
}else if(action.equals("addproduct")){//用户单击“添加”按钮
    String id=request.getParameter("productId");
    String name=request.getParameter("productName");
    String price=request.getParameter("productPrice");
    String num=request.getParameter("productNum");
```

```

String describe=request.getParameter("productDescribe");
int line=database.addProduct(id,name,price,num,describe);
if(line==1){//更新的记录数为 1, 则表明添加成功
    session.setAttribute("insert","ok");
    vec=database.getAllProduct();
    session.setAttribute("showresult",vec);
}else{
    session.setAttribute("insert","notok");
}
this.forward(request,response,"addproduct.jsp");
}else if(action.equals("showaddddelete")){//用户单击“查看已有商品”链接
    this.forward(request,response,"showaddddelete.jsp");
}
}

```

完成了上述步骤, 即完成了 Servlet 的开发。

18.5.4 开发显示与删除页面

本项目中的显示与删除页面为 showaddddelete.jsp, 其以分页的方式显示当前所有的商品, 并且可以在该页面中删除商品。按照以下步骤完成 showaddddelete.jsp 的开发。

(1) 向 Shop 中添加一个名称为 showaddddelete 的 JSP。

(2) 在编辑器窗口中打开 showaddddelete.jsp 文件, 删除其中自动生成的代码, 并将下列代码添加到其中:

```

<%@page contentType="text/html" autoFlush="true"%>
<%@page pageEncoding="gb2312"%>
<%@page import="java.sql.*"%>
<%@page import="java.util.*"%>
<%@page import="org.netbeans.Web.*"%>
    <%//声明两个变量
        int pageRecordNum=5;
        int pageCount;
    %>
    <%
        String tempPageNum=request.getParameter("pageno");
        try{
            if(tempPageNum==null){//在未使用上一页, 下一页之前, pageno 为 null
                pageCount=1;
            }else {
                pageCount=Integer.parseInt(tempPageNum);
            }
        }catch (Exception e){
            e.printStackTrace();
        }
    %>
<html>
<head>

```

```

<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>全部商品</title>
</head>
<body>
    <form action="AdminServlet" method="post" name="result">
        <input type="hidden" name="pageno" value="<%=pageCount%>" />
        <!-- pageno 用来记录当前的页数-->
        <%
            if(session.getAttribute("adminname")==null){
                session.setAttribute("notlogin","notlogin");
            }
            <jsp:forward page="login.jsp"/>
        <% }
            else if((session.getAttribute("showresult"))!=null){ //结果不为空
        <%
            <table align="center" width="100%" bgcolor="dddddd"
                border="1" cellpadding="0" cellspacing="0">
                <tr>
                    <td>商品编号</td>
                    <td>商品名称</td>
                    <td>商品价格</td>
                    <td>库存数量</td>
                    <td>商品描述</td>
                    <td>&nbsp;</td>
                </tr>
                <%
                    //从 session 中获取的 showresult 值作为结果集,
                    // 该属性的值在 SearchProductServlet 中设置
                    Vector vec=(Vector)session.getAttribute("showresult");
                    int size=vec.size(); //获取记录总数
                    for(int i=(pageCount-1)*5;i<(pageCount-1)*5+pageRe-
                        cordNum;i++){
                        ProductBean tempbean=(ProductBean)vec.elementAt(i);
                        int productId=tempbean.getProductId(); //获取 ID 属性的值
                    }
                <%
                <tr>
                    <td><%=productId%></td>
                    <td><%=tempbean.getProductName()%></td>
                    <td><%=tempbean.getProductPrice()%></td>
                    <td><%=tempbean.getProductNum()%></td>
                    <td><%=tempbean.getProductDescribe()%>&nbsp;</td>
                    <td><a href="AdminServlet?productId=<%=productId%>
                        &pageno=
                            <%=pageCount%>&action=delete">删除</a></td>
                    <!-- 在获得的 product_describe 的值后面添加&nbsp;,
                        这样即使 product_describe 为空也不影响显示效果-->
                </tr>

```

```

        <%
            if(i>=size-1)
                break;
        %>
    </table>
    <%
        if(pageCount!=1) {
    %>
    <a href="AdminServlet?action=previous&pageno=<%=pageCount-1%>">
    上一页</a>
    <% }

        int lastPageNum=size/pageRecordNum;
        if(size%pageRecordNum!=0) {
            lastPageNum++;
        } if (pageCount!=lastPageNum) {
    %>
    <a href="AdminServlet?action=next&pageno=<%=pageCount+1%>">下一页</a>
    <% }%>
    <br>共<%=lastPageNum%>页, 当前第<%=pageCount%>页<br>
    <a href="AdminServlet?action=add">添加新商品</a>
    <% } %>
</form>
</body>
</html>

```

提示 action 与 href 属性的值 AdminServlet 是该 JSP 页面中数据要提交到的 Servlet 的 URL 映射值。

完成了上述步骤, 就已经成功地完成了 showadddelete.jsp 的开发。

18.5.5 开发登录页面与添加商品页面

login.jsp 是管理员登录页面, addproduct.jsp 是添加商品页面, 按照如下步骤完成这两个 JSP 页面的开发。

- (1) 向 Shop 中添加一个名称为 login 的 JSP。
- (2) 在编辑器窗口中打开 login.jsp 文件, 删除其中自动生成的代码, 并将下列代码添加到其中:

```

<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
        <title>登录界面</title>
    </head>
    <body>

```



```

<form action="AdminServlet" name="myform" method="post">
<table align="center" width="100%" bgcolor="dddddd"
border="1" cellpadding="0" cellspacing="0">
<tr>
<td>账号: </td>
<td><input type="text" name="username" size="40"></td>
</tr>
<tr>
<td>密码: </td>
<td><input type="password" name="password" size="40"></td>
</tr>
<tr>
<td align="right"><input type="submit" value=" 提交 " ></td>
<td align="left"><input type="reset" value=" 重填 " ></td>
</tr>
</table>
</form>
</body>
<%
if(session.getAttribute("notlogin")!=null){
String templogin=(String)session.getAttribute("notlogin");
if(templogin.equals("loginfail")){
out.println("<hr>用户名或密码错误,请重新输入!");
}else if(templogin.equals("notlogin")){
out.println("<hr>你还没有登录,请先在此处登录!");
}
session.setAttribute("notlogin",null);
}else if(session.getAttribute("adminname")!=null){
String adminname=(String)session.getAttribute("adminname");
out.println("<hr>当前管理员: "+adminname);
}
%>
</html>

```

(3) 向 Shop 中添加一个名称为 addproduct 的 JSP。

(4) 在编辑器窗口中打开 addproduct.jsp 文件,删除其中自动生成的代码,并将下列代码添加到其中:

```

<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>添加商品</title>
</head>
<body>

```



```
<form action="AdminServlet" name="myform" method="post">
    商品编号: <input type="text" name="productId"><br>
    商品名称: <input type="text" name="productName"><br>
    商品价格: <input type="text" name="productPrice"><br>
    商品数量: <input type="text" name="productNum"><br>
    商品说明: <br><textarea cols="30" rows="5" name="productDescribe"></textarea><br>
    <input type="hidden" value="addproduct" name="action">
    <input type="submit" value="添加">
    <a href="AdminServlet?action=showadddelete">查看已有商品</a>
</form>
<%
    if(session.getAttribute("adminname")==null){
        session.setAttribute("notlogin","notlogin");
    %>
<jsp:forward page="login.jsp" />
<% }
    if(session.getAttribute("insert")!=null){
        String insert=(String)session.getAttribute("insert");
        if(insert.equals("ok")){
            out.println("成功添加商品");
        }else{
            out.println("添加商品失败");
        }
        session.setAttribute("insert",null);
    }
    %>
</body>
</html>
```

完成了上述步骤,就已经成功地完成了 login.jsp 与 addproduct.jsp 的开发。到此为止,与管理模块有关的组件就已经开发完成了。编译并部署该项目,在浏览器的地址栏中输入如下内容:

http://localhost:8877/Shop/login.jsp

则浏览器中会出现如图 18-12 所示内容。读者可以按照 18.2.3 节介绍的内容进行操作。

18.6 小结

本章首先对 MVC 架构进行了详细的介绍,然后结合普通电子商务网站的用户查询与购买以及管理部分,详细地介绍了如何使用 JSP、Servlet 与 JavaBean 开发一个具有 MVC 架构的购物网站。通过该项目可以看到,使用 JSP、Servlet 与 JavaBean 开发的具有 MVC 架构的项目具有易维护、易修改、模块之间的耦合性低等优点。在开发实际项目的时候,应该多使用这种架构。

本篇为 Java EE 篇，在本篇中结合 JBoss 详细地讲述了如何在 NetBeans 中进行企业级应用的开发，主要包括如下内容：

- Java EE 简介；
- 搭建 Java EE 开发环境；
- 有状态会话 Bean 的介绍与项目的开发；
- 无状态会话 Bean 的介绍与项目的开发；
- CMP 的介绍与项目的开发；
- 消息驱动 Bean 的介绍与项目的开发。

第 4 篇 Java EE 篇

第 19 章 NetBeans 与 Java EE

第 20 章 无状态会话 Bean——用户消费信息登记

第 21 章 有状态会话 Bean——实现购物车

第 22 章 开发 CMP 实体 Bean——图书信息管理

第 23 章 消息驱动 Bean——商品问题反馈系统

第 19 章

NetBeans 与 Java EE

NetBeans 在开发 Java EE 应用方面的能力非常强大，本章将向读者简要介绍 Java EE 以及 NetBeans 对 Java EE 开发各方面的支持。

19.1 Java EE 概述

Java EE 企业版开发平台定义了开发多层企业应用程序的标准，Java EE 将企业应用程序构建于标准化、模块化的组件之上，并为这些组件提供了完整的服务体系，还自动处理了应用程序中大量具体实现的细节，避免了复杂的编程工作，从而简化了企业应用程序的开发过程。

Java EE 的基石就是 Enterprise JavaBeans (EJB)，EJB 是建立基于 Java 的服务器端组件的标准，其定义了如何编写服务器端组件，提供了组件与管理组件的应用服务器之间的标准约定。EJB 是一种组件架构，使得开发人员能够快速开发出具有伸缩性的企业级应用。在下一节中将对 EJB 做详细的介绍。

提示 Java EE 实际上就是 J2EE，Sun 公司决定从 J2EE 1.5 版本开始把 J2EE 改名为 Java EE，所以实际上 Java EE5 就是 J2EE 1.5。

通过使用 Java EE 可以带来如下优点。

- 自由选择：J2EE 不限制在某一家厂商，同时支持跨平台开发。这就鼓励厂商竞争，生产出更好的产品。
- 快速开发和部署：J2EE 显著降低了开发和发布多层解决方案的成本和复杂度，能够实现快速开发企业级应用程序。

19.2 Enterprise JavaBeans 概述

Enterprise JavaBeans (EJB) 是 Java EE 的核心内容，是建立基于 Java 的服务器端组件的标准。本节将会对 EJB 进行简单的介绍，主要包括如下内容：

- EJB 简介;
- EJB 的优点及适用场合。

19.2.1 EJB 简介

本小节对 EJB 进行了简单的介绍,主要包括了如下几点内容:

- EJB 的定义;
- EJB 的工作框架;
- EJB 的分类。

下面将分别对这几部分内容进行介绍。

1. EJB 的定义

在 EJB 规范中定义了如下内容:“EJB 是一种基于组件的分布式计算结构。EJB 是分布式地面向事务的企业应用程序组件。”简单来说,EJB 是服务器端的组件,它包括了应用程序中的业务逻辑。业务逻辑是指实现应用程序领域功能的代码程序。例如,在 B2C 的电子商务中,EJB 可以调用 `searchProduct` 和 `makeOrder` 方法完成业务逻辑。通过启用这些方法,远程客户能够访问提供的在线下单购物服务。

EJB 经常与企业信息系统(Enterprise Information System, EIS)交互,如数据库、遗留系统以及其他的 EJB。在同一时间不同类型客户的访问都需要 EJB 的服务,这些客户包括其他的 EJB、Web 应用以及应用客户。

2. EJB 的工作框架

在运行过程中,EJB 处于一个 EJB 容器(container)中。EJB 容器提供 EJB 的部署和运行环境,包括为 EJB 提供各种服务,如安全、事务、部署和并发管理。EJB 的工作框架如图 19-1 所示。

在 EJB 容器中安装 EJB 的过程称为 EJB 的部署(deployment),EJB 容器是应用服务的部分。由于 EJB 是方便简捷的组件,因此,可以付出很少的劳动,用现有的 EJB 创建新的应用组合程序。

3. EJB 的分类

EJB 2.0 规范指出了如下几种类型的 EJB。

- 实体 Bean,包含容器管理持久性实体 Bean 与 Bean 管理持久性实体 Bean。
- 会话 Bean,包含有状态会话 Bean 和无状态会话 Bean。
- 消息驱动 Bean。

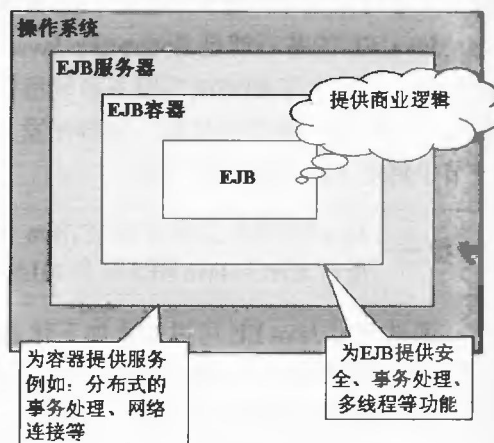


图 19-1 EJB 的工作框架



说明

不同类型的 EJB,具有不同的功能及使用方法,在后面的章节中,将会对这几种 EJB 进行详细的介绍。

图 19-2 给出了这几种 EJB 的分类图。

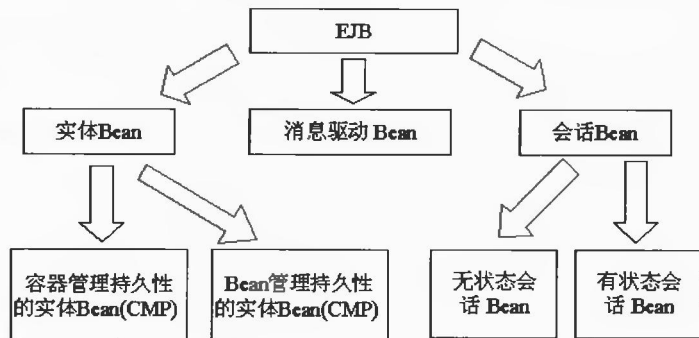


图 19-2 EJB 的分类

19.2.2 EJB 的优点及适用场合

EJB 的出现大大简化了大型分布式应用程序的开发，其原因主要有以下两点。

- EJB 容器为 EJB 提供系统级的服务，其主要集中于业务逻辑的开发，并且应用的业务逻辑包含在 EJB 中，而不是在客户中，所以客户端开发人员可以集中精力开发表示层的逻辑。
- 客户端开发人员没有必要编写实现业务规则或访问数据库的程序。因此，客户端可以是现在流行的瘦客户端，这个特性对于小型设备的用户尤其有益，如手机和 PDA。

EJB 主要适合有以下特点和需要的应用程序。

- 有升级需要的：要适应逐步增长的用户数，一种方法就是需要跨越许多机器分布应用程序的各个组成部分。一个应用中的 EJB 不仅可以运行在不同的机器上，而且其位置对用户可以是透明的。
- 面向事务的：EJB 支持通过容器服务的事务处理方式，这个机制可以管理共享对象的并发访问，确保数据的完整性。
- 多种客户类型的：只要使用很少的代码远程客户就可以非常方便地访问 EJB，这些客户可以是瘦客户，也可以是其他各种类型的客户端。

19.3 NetBeans 对 Java EE 的支持

NetBeans 对 Java EE 的开发者提供全面的支持。NetBeans 的高级向导可以引导开发者创建完整的 Java EE 应用程序和独立的 Java EE 组件，如 Web 应用程序、Servlets、Java Server Faces 应用程序、企业 Java 可重构组件模块 (EJB 模块)、企业 Java 组件 (enterprise beans) 以及 Web 服务等。另外 NetBeans 还提供了绑定 SJS 应用服务器 9.0 版本的完整运行环境。

使用 NetBeans 配合相应的 Java EE 应用服务器不仅可以方便地开发、生成企业级应用程序，而且还可以同时在 IDE 中进行部署和调试等操作，这给企业级应用程序的开发人员提供了极大的方便。

提示

SJS 是 Sun Java System Application Server 的简称, 它是由 Sun 公司开发的 Java EE 应用服务器。SJS 应用服务器是与 Java EE 开发平台兼容的参考技术实现。其既可以单独下载使用, 也可以和 NetBeans 绑定下载。读者如果有兴趣可以到 Sun 的官方网站 <http://www.sun.com.cn> 进行下载。

为了充分利用 NetBeans 中与 Java EE 应用开发相关的功能, 需要对 NetBeans 的开发环境进行相应的配置, 在本章中的后继部分主要向读者介绍如何在 NetBeans 中连接及管理目前主流的数据库以及 Java EE 应用服务器, 如 MySQL、Oracle、WebLogic、JBoss 等。

19.4 MySQL 数据库

NetBeans 支持连接各种主流的数据库系统。本节将介绍 NetBeans 连接 MySQL 数据库的方法和步骤。主要包括如下内容:

- 安装 MySQL 数据库;
- 配置 MySQL 数据库;
- NetBeans 连接 MySQL。

19.4.1 安装 MySQL 数据库

要想在 NetBeans 中连接 MySQL 数据库, 首先要对其进行安装。按照如下步骤完成 MySQL 的安装。

(1) MySQL 的安装程序是一个名为 Setup 的可执行文件, 双击该文件, 经过一段时间的初始化工作以后, 出现如图 19-3 所示的欢迎界面。

(2) 在如图 19-3 所示的窗口中单击“Next”按钮继续安装, 此时界面如图 19-4 所示。

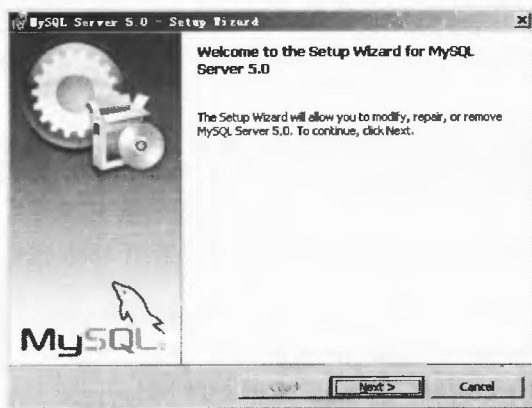


图 19-3 欢迎界面

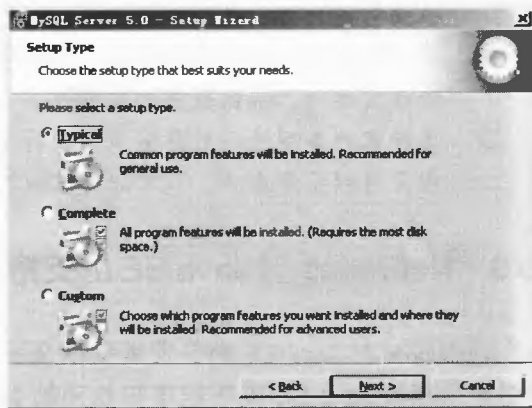


图 19-4 选择安装类型

(3) 可以在如图 19-4 所示的窗口中选择安装的类型, 包括 Typical (典型安装)、Complete (完全安装) 和 Custom (自定义安装)。在此选择自定义安装。单击“Next”按钮继续安装。此时界面如图 19-5 所示。

(4) 可以在如图 19-5 所示窗口中设置要安装的组件与安装目录。在此仅更改安装目录, 而其余选项均使用默认值。单击“Next”按钮继续安装, 此时界面如图 19-6 所示。

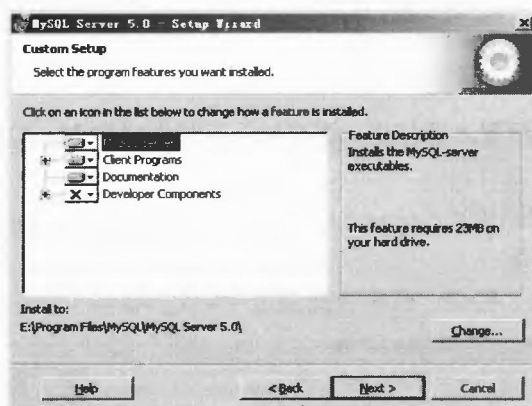


图 19-5 自定义安装界面

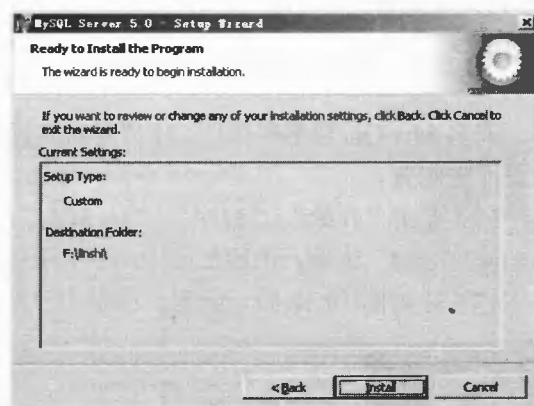


图 19-6 确认安装信息界面

(5) 在如图 19-6 所示的窗口中给出了在前面几步中设置的信息, 包括安装类型和安装目录。如果需要重新设置这些内容, 可以单击“Back”按钮。如果这些信息确认无误, 单击“Install”按钮开始安装。此时界面如图 19-7 所示。

(6) 在如图 19-7 所示的窗口显示了安装进度。经过一段时间的安裝后, 出现如图 19-8 所示界面。

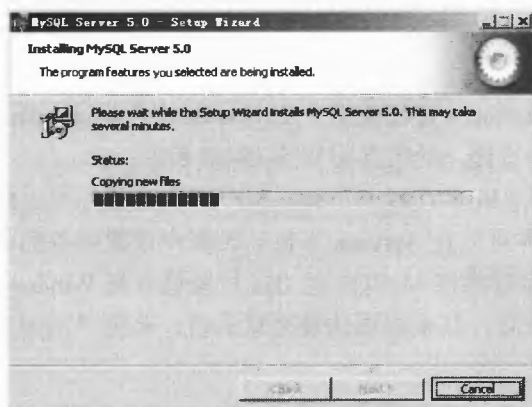


图 19-7 安装界面

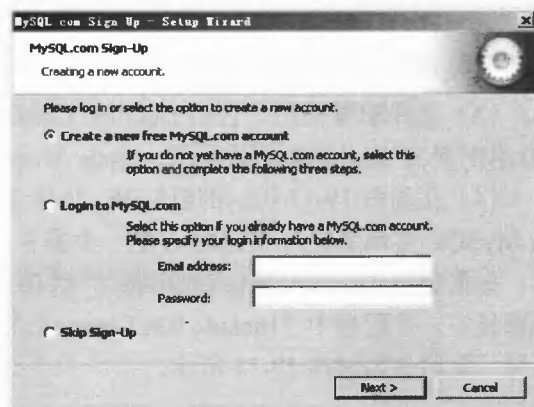


图 19-8 创建账号

(7) 可以在如图 19-8 所示的窗口中选择创建账号的方式, 包括创建免费账号(Create a new free...), 使用已有的账号(Login to MySql.com)和不创建账号(Skip Sign-Up)。默认选项为“创建免费账号”, 在此选择 Skip Sign-Up 选项, 单击“Next”按钮, 此时界面如图 19-9 所示。

(8) 将“Configure the MySql Server now”设置为非选中状态, 单击“Finish”按钮完成。

完成了上述步骤, 就已经成功地完成了



图 19-9 安装完成界面

MySQL 的安装。

19.4.2 配置 MySQL 数据库

安装 MySQL 数据库以后,还需要对其进行配置才能使用,按照如下步骤完成对 MySQL 数据库的配置。

(1) 单击“开始”/“程序”/“MySQL”/“MySQL Server 5.0”/“MySQL Server Instance Config Wizard”选项,出现如图 19-10 所示的界面。

(2) 在如图 19-10 所示的窗口中单击“Next”按钮,此时界面如图 19-11 所示。



图 19-10 配置 MySQL 的欢迎界面

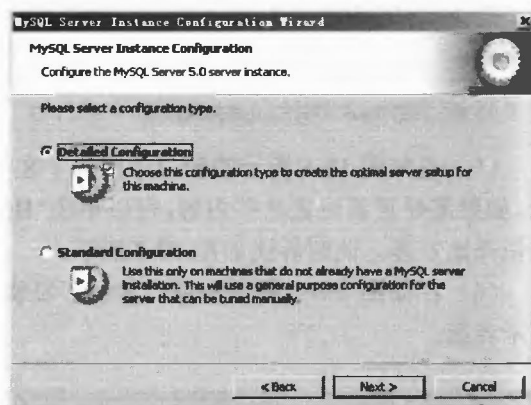


图 19-11 选择配置类型

(3) 选择配置类型,包括 Detailed Configuration (详细配置)与 Standard Configuration (标准配置)。在此选则标准配置。单击“Next”按钮,此时界面如图 19-12 所示。

(4) 在如图 19-12 所示的窗口中,如果选中“Install As Windows Service”选项,可以将该 MySQL 实例绑定为 Windows 的一个服务,并可以在 Service 下拉列表框中设置服务的名称。如果选中“Include Bin Director...”选项,可以将该 MySQL 的 Bin 目录包含到 Windows 的路径中。在此选中“Include Bin Director...”选项,其余选项均使用默认值。单击“Next”按钮,此时界面如图 19-13 所示。

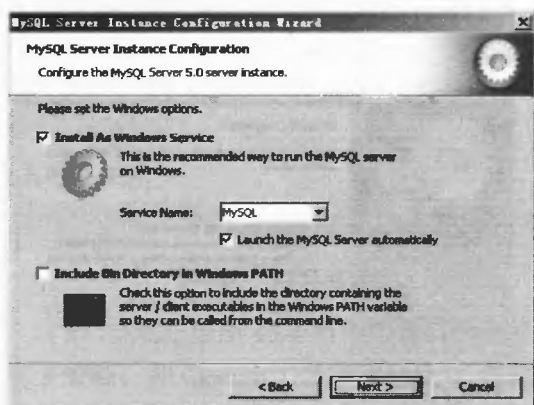


图 19-12 配置 MySQL 实例

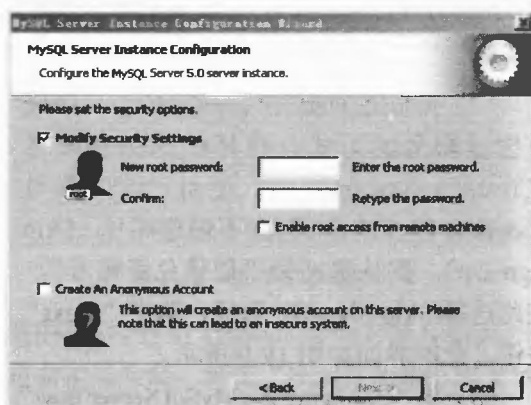


图 19-13 设置安全选项

(5) 可以在如图 19-13 所示的窗口中为 root 设置密码,还可以在此窗口中创建账户。在

此仅仅为 root 设置密码。设置完成以后，单击“Next”按钮，此时界面如图 19-14 所示。

(6) 在如图 19-14 所示的窗口中列出了将要配置的一些选项，单击“Execute”按钮开始配置。经过一段时间的配置以后，界面如图 19-15 所示。

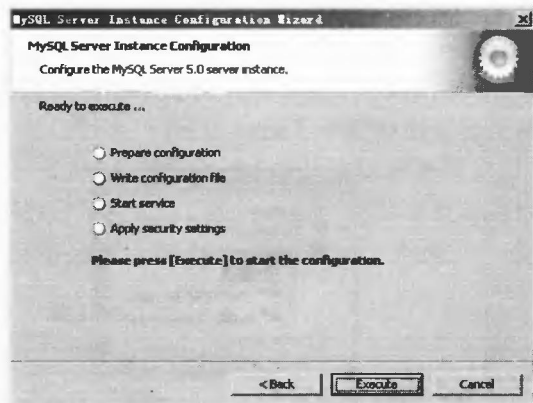


图 19-14 处理配置界面

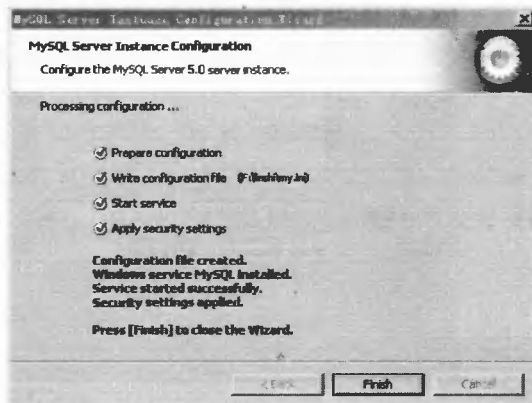


图 19-15 配置完成界面

(7) 在如图 19-15 所示的窗口中单击“Finish”按钮完成。

19.4.3 连接 MySQL 数据库

在 NetBeans 中连接 MySQL 数据库的步骤如下。

(1) 选中主窗口中的“Run Time”窗口，展开窗口中的 Databases 节点，在此节点下会发现 Drivers 子节点，如图 19-16 所示。

(2) 用鼠标右键单击“Drivers”节点，在弹出的菜单中选择“Add Driver”选项，将弹出添加 JDBC 驱动的向导窗口，如图 19-17 所示。



图 19-16 Drivers 子节点的位置

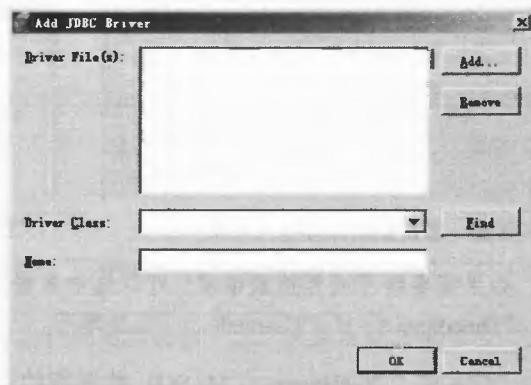


图 19-17 添加 JDBC 驱动的向导窗口

(3) 单击窗口中的“Add...”按钮，找到存放在磁盘上的 MySQL JDBC 驱动程序文件，这时就会发现“Driver File(s)”列表框中出现了新添加的驱动程序包，如图 19-18 所示。



提示

MySQL 数据库的安装包中并不包含相应的 JDBC 驱动，如果需要可以到 MySQL 的官方网站 <http://www.mysql.com/> 下载。

(4) 在“Driver Class”下拉列表框中选择需要的驱动程序类,然后单击“OK”按钮完成JDBC 驱动程序的添加。这时展开“Run Time”窗口中的 Drivers 节点,如果添加成功将出现代表 MySQL JDBC 驱动的子节点,如图 19-19 所示。

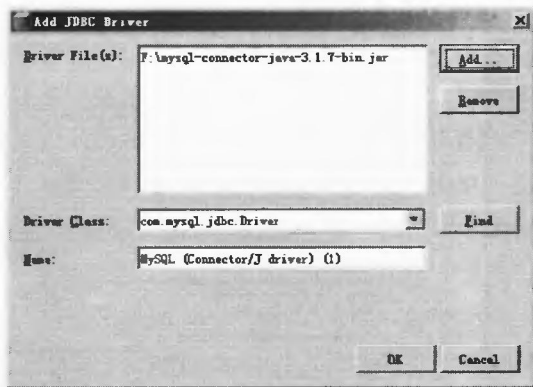


图 19-18 在向导窗口中添加驱动程序包

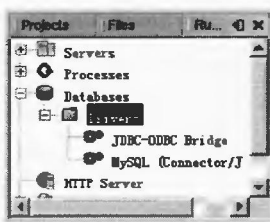


图 19-19 MySQL JDBC 驱动的子节点

(5) 用鼠标右键单击“MySQL (Connector/J driver)”节点,在弹出的菜单中选择“Connect Using...”选项,这时会弹出“New Database Connection”窗口,如图 19-20 所示。

(6) 在“Database URL”字段中指定要连接的 MySQL 数据库的主机名、端口号和数据库名,如 jdbc:mysql://localhost:3306/test。

(7) 在 User Name 和 Password 字段中分别给出指定数据库的用户名和密码,单击“OK”按钮,完成数据库的连接。连接成功后会发现“Databases”节点下会多出一个数据库连接的子节点,如图 19-21 所示。

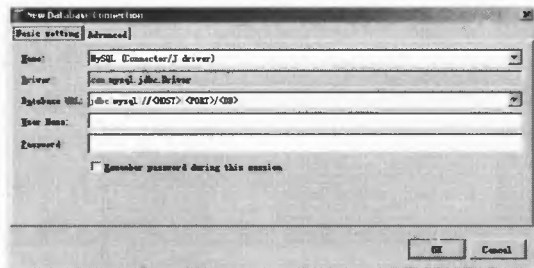


图 19-20 新建数据库连接向导窗口

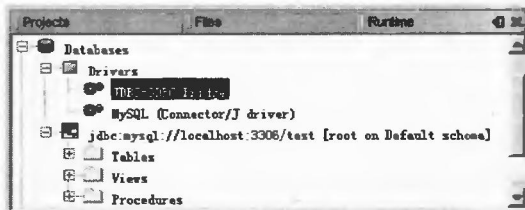


图 19-21 数据库连接成功后的“Runtime”窗口

提示 如果需要断开或连接数据库,可以选中数据库的连接节点,在鼠标的右键菜单中使用“Disconnect”或“Connect...”项目即可。

至此已经完成了 NetBeans 与 MySQL 数据库的连接,可以看到在数据库连接节点下分别有 Tables、Views 以及 Procedures 3 个子节点,它们分别表示数据库中的表、视图与存储过程。在下一节中将介绍如何通过这 3 个节点来管理 MySQL 数据库。

19.4.4 管理 MySQL 数据库

在 NetBeans 中对 MySQL 数据库的管理主要包括对表中各项信息的管理,下面将分别进行介绍。

1. 用 NetBeans 创建表

在 NetBeans 中可以轻松通过鼠标操作在数据库中创建新表，免去了编写 SQL 脚本的麻烦，同时它让不熟悉 SQL 的读者也可以方便地操作。创建表的具体步骤如下。

(1) 用鼠标右键单击“Tables”节点，选择右键菜单中的“Create Table...”选项，系统将弹出用于创建表的向导窗口，如图 19-22 所示。

(2) 在“Table name”字段中指定要创建的新表的名称，如 MyNewTable。

(3) 在向导窗体中间的表格中系统已经默认给表创建了一个字段，可以在 Column name 中指定字段名称，如 MyID，同时可以通过勾选 Key、Index、Null、Unique 选项来设置字段对应的属性，具体含义如表 19-1 所示。

表 19-1 各个选项的具体含义

项 目	含 义
Key	用来确定对应字段是否是键值字段
Index	用来确定对应字段元是否使用索引
Null	用来确定对应字段是否允许为空值
Unique	用来确定对应字段是否使用惟一性约束

(4) 通过 Data type 的下拉列表可以选择字段元所需要的类型，如图 19-23 所示。

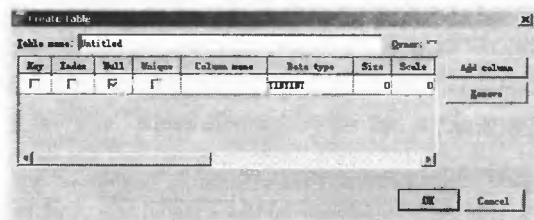


图 19-22 创建表的向导窗口

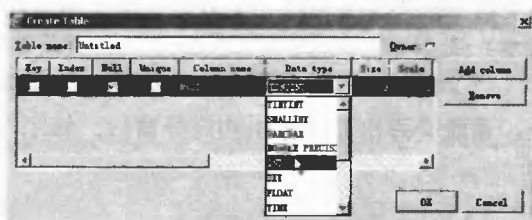


图 19-23 选择表字段类型

(5) 在“Size”和“Scale”中可以分别指定字段的大小和范围。如果需要增加新的字段元可以单击“Add column”按钮，这样在表格中就会出现一个新建行（代表一个数据库字段），对新字段的设置同上面的步骤。

(6) 所有的字段都添加完毕后，单击“OK”按钮完成数据库表的创建。创建成功后在 Tables 节点下出现了一个 MyNewTable 节点，即刚刚创建的数据库表。



提示

如果需要删除一张表，则用鼠标选中要删除的表节点，选择右键菜单中的“Delete”选项即可完成表的删除。

2. 在已有的表中增加字段

在很多情况下有可能需要在已有的表中添加新的字段，用 NetBeans 也可以很方便地实现这一功能，具体步骤如下。

(1) 用鼠标右键单击需要添加字段的表节点，如 MyNewTable 节点，选择右键菜单中的“Add Column...”选项，系统会弹出添加字段向导窗口，如图 19-24 所示。

(2) 在“Name”字段中指定要添加字段的名称,在“Type”下拉列表框中选择适当的字段类型,在 Size、Scale 和 Default 字段中分别指定字段的大小,范围和默认值。

(3) 通过勾选“Null”、“Unique”选项确定字段的空值约束和惟一性约束。

(4) 如果希望对新字段元使用索引可以选中“Index”复选框,并在其后的下拉列表中选择恰当的索引类型。

(5) 选中“Check”复选框,在后面的文本框中可以编写字段特殊约束的脚本,如 `MySalary<10000`。

(6) 完成对字段的各项设置后,单击“OK”按钮结束新字段的添加。添加成功后展开 `MyNewTable` 节点,下面的 `MySalary` 子节点,即添加的新字段。

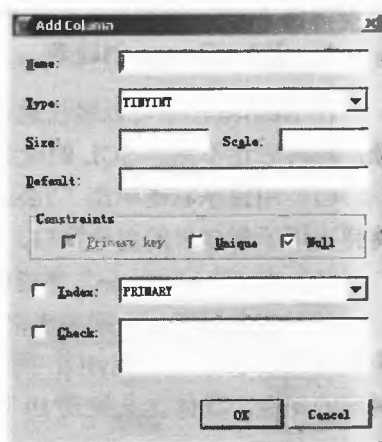


图 19-24 添加字段向导窗口

提示 如果需要删除表中的某个字段,则用鼠标在要删除的字段节点上单击鼠标右键,选择右键菜单中的“Delete”选项即可完成字段的删除。

3. 给已有的表添加索引

通过 NetBeans 可以很方便地对表中的索引进行管理,当展开一个表节点后会发现下面有一个 `Indexes` 子节点,此节点代表的就是表中的索引;展开它可以发现在其子节点中列出了表中所有的索引;展开一个索引节点可以看到下面列出了索引所关联的各个字段,如图 19-25 所示。用 NetBeans 向表中添加索引的步骤如下。

(1) 用鼠标右键单击某一个表的“Indexes”节点,选择右键菜单中的“Add index...”选项,系统会弹出添加索引的向导窗口,如图 19-26 所示。

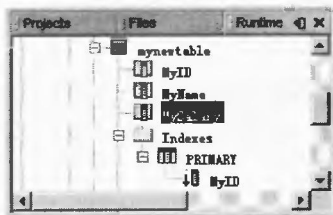


图 19-25 表中的索引

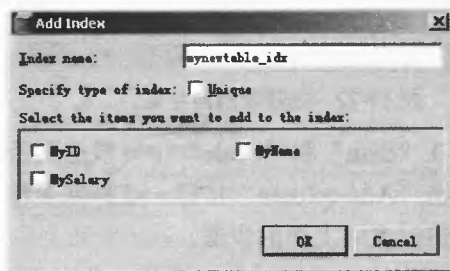


图 19-26 添加索引的向导窗口

(2) 在“Index name”字段元中指定添加索引的名称,如 `MyIndex`;再通过“Unique”复选框确定是否使用惟一性索引。

(3) 勾选索引所涉及的字段元,单击“OK”按钮,完成索引的添加。添加成功后在 `Indexes` 节点下有 `MyIndex` 节点,即新添加的索引。

提示 如果需要删除索引,则在要删除的索引节点上单击鼠标右键,选择右键菜单中的“Delete”选项,即可完成索引的删除。

4. 管理表中的资料

通过 NetBeans 不但可以方便地管理表的结构,对表中的资料也可以很容易地进行管理。

在一个表节点上单击鼠标右键，选择右键菜单中的“View Data...”选项，系统会出现表资料的管理窗口，如图 19-27 所示。

在接口的下方是一个表格，列出了表中的资料，可以在接口上方的文本区域中输入要执行的 SQL 语句对表中的内容进行“增”、“删”、“改”、“查”等方面的管理。具体操作步骤如下。

(1) 在 SQL 代码区域中输入要执行的 SQL 语句，如：

```
insert into payment_table values ('Test','credit','9999',99.88);
```

(2) 单击  按钮执行此 SQL 语句，接口下方的区域将显示执行的结果，如图 19-28 所示。

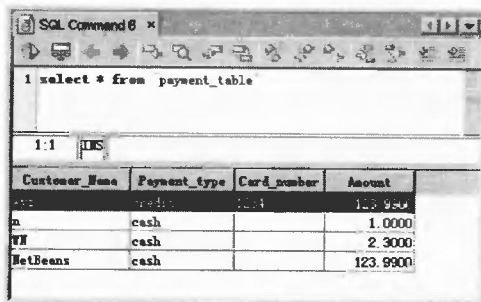


图 19-27 表资料的管理窗口

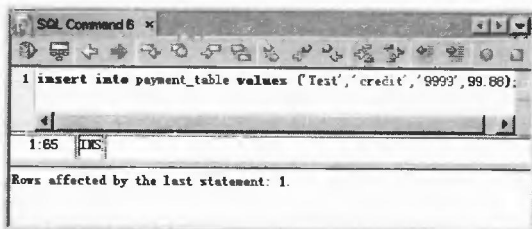


图 19-28 用 SQL 对表内容进行管理

如果执行的是检索性 SQL 语句，则会在接口的下方显示检索的结果，也就是一个包含检索结果表格；如果执行的是非检索性 SQL 语句，则会显示语句执行的情况，如“Rows affected by the last statement: 1.”。

19.5 在 NetBeans 中连接 Oracle10g

前面已经介绍了 NetBeans 与 MySQL 数据库的连接以及使用知识，下面将介绍 NetBeans 中连接和使用 Oracle10g 的方法和步骤。

19.5.1 安装 Oracle10g

在 NetBeans 中连接 Oracle10g 之前，首先要确保机器上已经成功地安装配置了 Oracle10g。在本小节中，将为读者介绍安装 Oracle10g 的方法。按照如下步骤完成 Oracle10g 的安装：

提示 如果读者现在还没有 Oracle10g，可以到 Oracle 的官方网站 <http://www.oracle.com> 下载，从 Oracle10g 开始，个人用户可以免费下载使用。

(1) Windows 版 Oracle10g 的安装程序中包含一个名称 Setup 的可执行文件。双击该文件，经过一段时间的初始化工作以后，出现如图 19-29 所示界面：

(2) 可以在图 19-29 所示窗口中选择安装方法，在此选择基本安装。此时可以设置 Oracle 的安装目录，安装类型，全局数据库名称以及数据库口令。安装类型有三种：企业版，个人

版, 标准版, 在此将安装类型设置为标准版。将全局数据库名称设置为 wyf, 并设置相应的口令。单击“下一步”按钮继续安装, 出现如图 19-30 所示的准备安装界面。

提示 在此处设置的数据库口令适用于 SYS, SYSTEM, SYSMAN 以及 DBSNMP 账户。

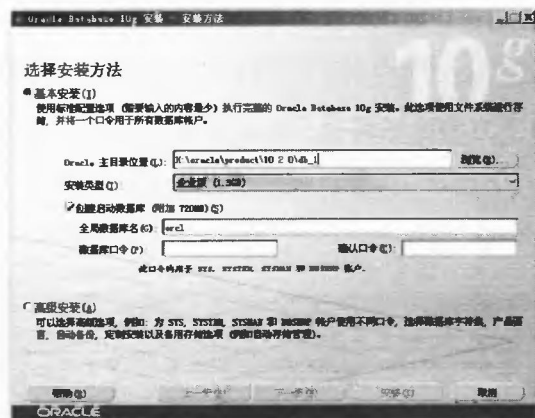


图 19-29 选择安装方式

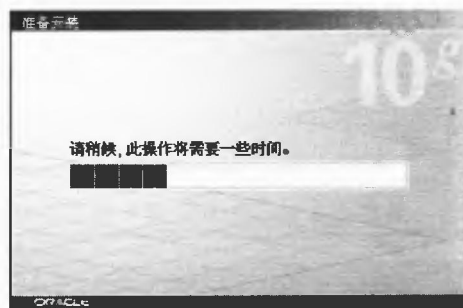


图 19-30 准备安装界面

(3) 该界面会存在一段时间, 时间的长短与机器配置有关。准备工作完成之后, 出现如图 19-31 所示界面。

(4) 图 19-31 所示窗口用来检查本机环境是否符合安装和配置所要安装的产品的所有最低要求。如果不能通过检测, 则不能继续安装, 通过检测后, 单击下一步继续安装, 此时界面如图 19-32 所示。

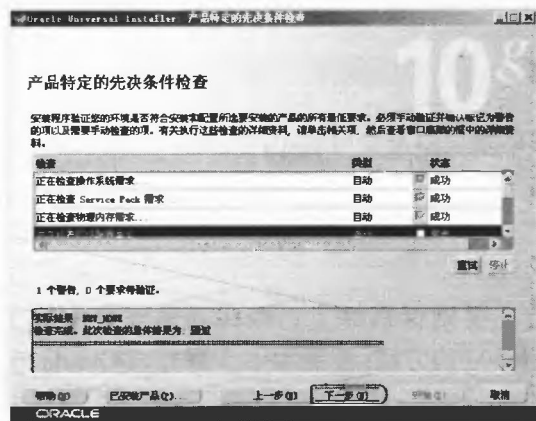


图 19-31 产品特定的先决条件检查



图 19-32 安装信息概要

(5) 图 19-32 所示窗口给出了一些安装信息的概要, 如全局设置、产品语言、空间要求等内容。确认这些信息准确无误后, 单击“安装”按钮开始安装。此时界面如图 19-33 所示。

(6) 根据机器配置不同, 该过程可能会持续一段很长的时间。其中进度条显示了安装进度。



图 19-33 安装界面

19.5.2 配置 Oracle10g

前面为读者讲述了与安装有关的内容，在本小节中，将为读者介绍与配置有关的内容。按照如下步骤完成配置。

(1) 在 Oracle10g 中，图 19-33 所示的安装完成后会自动进入配置界面。

(2) 在如图 19-34 所示的窗口中，配置向导将会自动使用相关工具配置并启动前面所安装的所有组件。首先使用“Oracle Net Configuration Assistant”工具配置与网络有关的内容。之后使用“Oracle Database Configuration Assistant”工具进行与数据库有关的配置，此时会出现如图 19-35 所示的窗口。



图 19-34 配置界面



图 19-35 配置数据库选项

(3) 图 19-35 所示窗口列出了要配置的内容、当前正在配置的内容以及进度。该配置过程可能会持续很长的一段时间。配置工作完成以后，出现图 19-36 所示窗口。

(4) 图 19-36 所示窗口给出了已经配置的数据库的相关信息。单击“确定”按钮，回到如图 19-34 所示的窗口。

(5) 此时配置向导会使用“iSQL*Plus Configuration Assistant”工具进行相关配置。配置

完成后,单击“下一步”按钮,出现如图 19-37 所示的窗口。

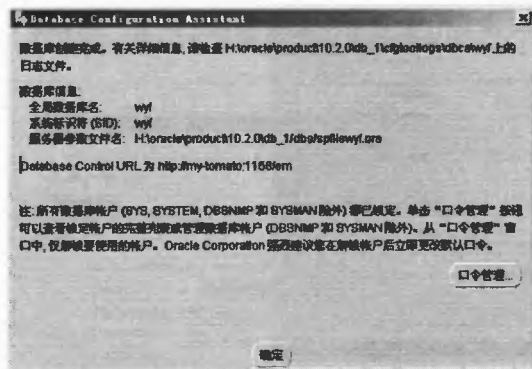


图 19-36 数据库配置信息



图 19-37 安装完成窗口

(6) 图 19-37 所示窗口给出了一些管理页面的 URL 值,用户可以记下这些值,以方便进入相应的管理页面进行管理。单击“退出”按钮退出安装程序。

19.5.3 测试 Oracle10g

安装与配置工作完成以后,还需要测试其是否可用,按照如下步骤完成其测试。

(1) 单击“开始”/“程序”/“Oracle-OraDb10g_home1”/“应用程序开发”/“SQL Plus”选项,出现如图 19-38 所示的窗口。

(2) 在如图 19-38 所示的窗口中输入相应的用户名和密码,单击“确定”按钮。出现图 19-39 所示窗口。



图 19-38 验证用户名和密码界面

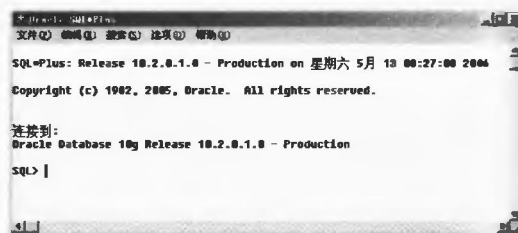


图 19-39 连接成功界面

如果能够出现图 19-39 所示的窗口,表明已经连接到 Oracle 数据库,即前面的安装与配置都是正确的。

19.5.4 NetBeans 连接 Oracle10g

在 NetBeans 中连接 Oracle 数据库的方法与连接 MySQL 数据库完全相同,只是使用的数

数据库驱动程序包根据 JDBC API 版本的不同分别为 classes11.jar 或 classes12.jar, 具体的步骤这里就不再赘述, 读者可以参照前面的章节自行完成, 连接成功后可以看到在 “Databases” 节点下出现了代表 Oracle 数据库连接的子节点, 如图 19-40 所示。

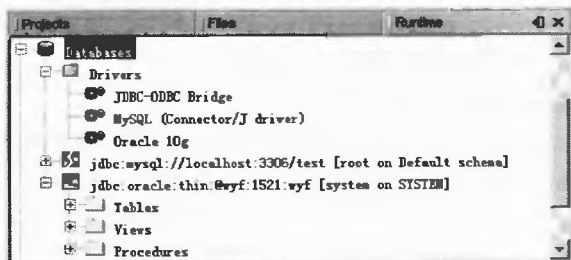


图 19-40 Oracle 数据库连接的子节点



提示

Oracle 与 MySQL 不同, 安装之后就已经有了 JDBC 驱动程序包, 不需要另外安装。JDBC 驱动程序包的位置在 Oracle 安装目录下 jdbc 子目录下的 lib 目录中; 其中 classes11.jar 是 JDBC 1.0 的驱动程序, classes12.jar 是 JDBC 2.0 的驱动程序, 读者可以根据需要选择适当版本的 JDBC 驱动。

19.5.5 管理 Oracle 数据库

NetBeans 对数据库的管理主要涉及表、视图、存储过程 3 个方面, 前面在介绍 NetBeans 与 MySQL 数据库时已经讲了对表的管理, 在这里就不再赘述。由于 MySQL 数据库对视图和存储过程支持很有限, 而 Oracle 在这些方面的功能都很强大, 因此这里主要对视图和存储过程的管理进行介绍。

1. 用 NetBeans 创建视图

视图是数据库三级模式结构中非常重要的一层, 在 NetBeans 中也可以像创建表一样轻松地通过鼠标操作在数据库中创建新的视图, 这对不熟悉 SQL 的读者是非常方便的一个功能。创建视图的具体步骤如下。

(1) 用鼠标右键单击 Oracle 数据库连接节点下的 Views 子节点, 选择右键菜单的 “Add View...” 选项, 系统将弹出创建视图的向导窗口, 如图 19-41 所示。

(2) 在 “View name” 字段中指定要创建视图的名称, 如 “view_student”。

(3) 在下面的文本区中输入用来创建视图的 SQL select 语句, 代码如下:

```
select * from student
```

- student 是数据库中已经存在的一张表。
- select 语句的结尾不可以有 “;” 号, 否则就会报错。

(4) 单击 “OK” 按钮完成视图的创建, 创建成功后在 Views 节点下将会出现一个 view_student 子节点, 即刚刚创建的视图。

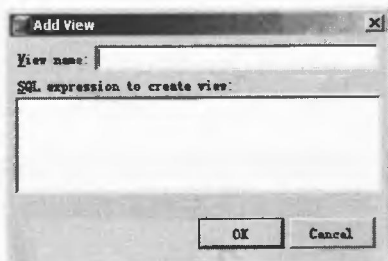


图 19-41 创建视图的向导窗口

提示 如果需要删除一个视图,则用鼠标选中要删除的视图节点,选择右键菜单中的“Delete”选项即可完成表的删除。

2. 管理视图中的资料

用 NetBeans 不但可以方便地创建视图,而且可以查看视图中的资料。选中要查看的视图节点,如 view_student 节点,单击鼠标右键,在弹出的菜单中选择“View Data...”选项,系统会自动弹出资料查看的窗口,如图 19-42 所示。

在窗口的上方可以输入对视图进行检索的 SQL 语句,窗口的下方由一个表格来显示检索的结果。

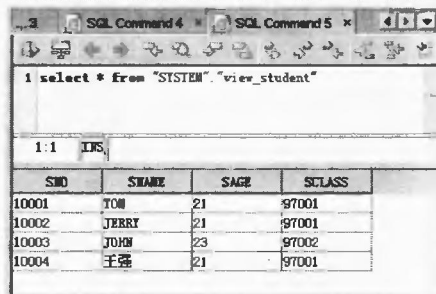


图 19-42 视图资料查看窗口

提示 在默认情况下将自动出现对视图所有内容进行检索的 SQL Select 语句,而下方的表格中将显示视图中所有的资料。

19.6 在 NetBeans 中配置 Weblogic 9.0

NetBeans 可以非常方便地和现在主流的 Java EE 应用服务器进行联合开发,在开发之前首先要将 NetBeans 和目标 Java EE 应用服务器进行连接。在本节中将介绍如何安装与配置 Weblogic 9.0, NetBeans 和 WebLogic 9.0 如何进行连接,以及如何在 NetBeans 中使用 WebLogic 9.0。

19.6.1 设置 Weblogic 9.0 的安装类型与目录

本小节中将介绍如何安装 Weblogic 9.0,在安装之前,首先要获得其安装文件。Windows 下的 Weblogic 9.0 安装文件为一个 .exe 格式的可执行文件,可以从 www.bea.com.cn 下载该安装文件。然后按照如下步骤完成其安装。

- (1) 双击该安装文件,初始化界面如图 19-43 所示。
- (2) 经过一段时间的初始化以后,出现如图 19-44 所示的欢迎界面。



图 19-43 初始化界面

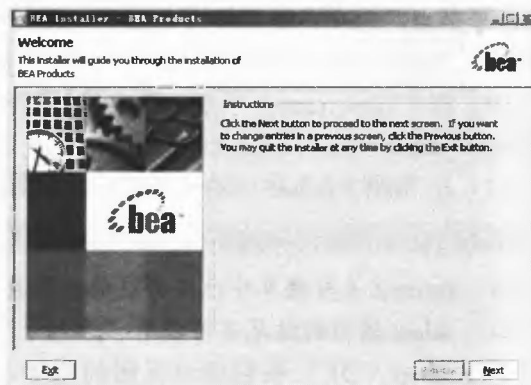


图 19-44 欢迎界面

(3) 在如图 19-44 所示的窗口中单击“Next”按钮, 打开如图 19-45 所示的界面。

(4) 图 19-45 所示窗口列出了安装 BEA 产品必须遵守的许可协议。选中“Yes”单选按钮后, 单击“Next”按钮, 此时界面如图 19-46 所示。

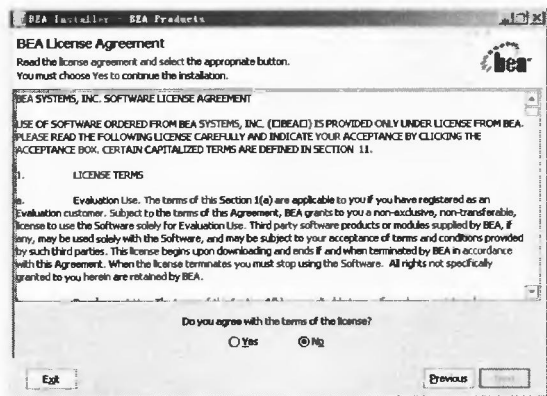


图 19-45 接受许可协议

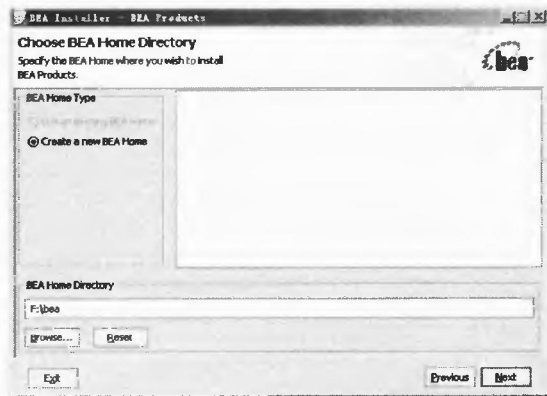


图 19-46 设置 BEA 产品的安装主目录

(5) 可以在如图 19-46 所示的窗口中设置安装 BEA 产品的主目录, 设置完成以后, 单击“Next”按钮。此时界面如图 19-47 所示。

(6) 可以在如图 19-47 所示窗口中设置安装类型, 即 Complete(完全安装)与 Custom(自定义安装)。在此选定完全安装。单击“Next”按钮, 此时界面如图 19-48 所示。

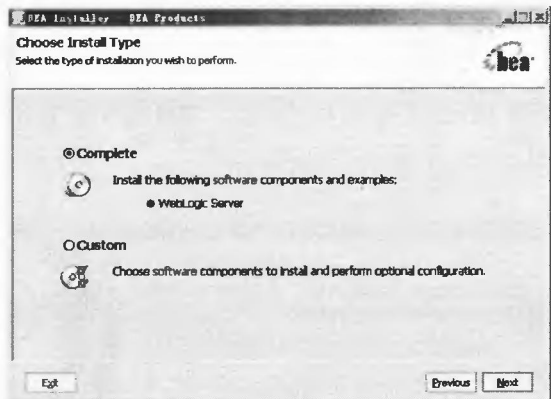


图 19-47 设置安装类型

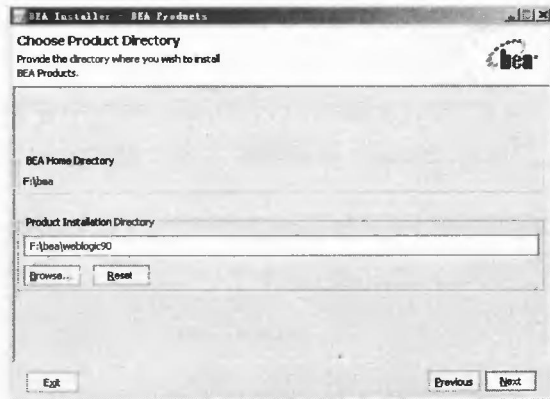


图 19-48 设置安装 Weblogic 9.0 的目录

(7) 在图 19-48 中设置安装 Weblogic 9.0 的目录。

19.6.2 设置其他安装选项并安装 Weblogic 9.0

上个小节设置了 Weblogic 9.0 的安装类型与目录, 在本小节中, 将继续设置其他的安装选项并安装 Weblogic 9.0。

(1) 在如图 19-48 所示的窗口中设置安装 Weblogic 9.0 的目录后, 单击“Next”按钮继续安装。此时界面如图 19-49 所示。

(2) 在如图 19-49 所示的窗口中, 设置在当前用户的开始菜单中创建 BEA 产品的快捷方

式，或者在所有用户的开始菜单中创建创建 BEA 产品的快捷方式。在此选择“**All UsersStartMenu folder**”选项（在所有用户的开始菜单中创建 BEA 产品的快捷方式），单击“**Next**”按钮进行安装。此时界面如图 19-50 所示。

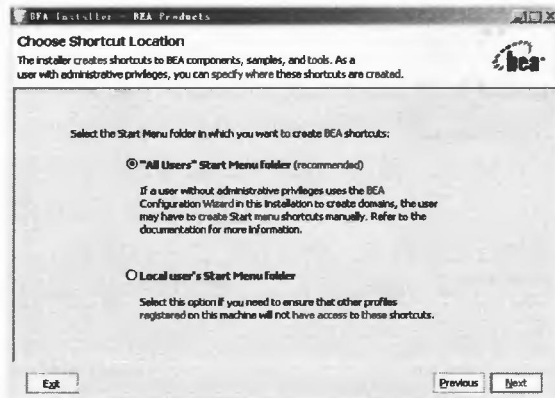


图 19-49 设置开始菜单选项

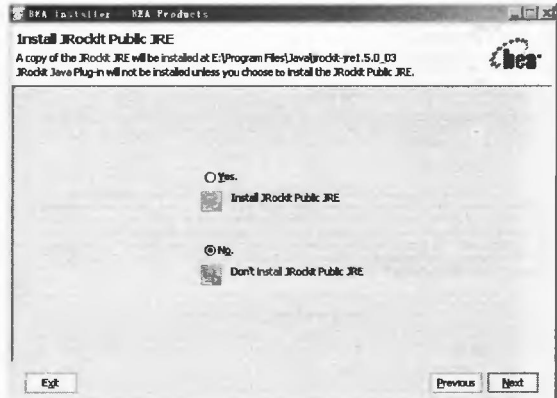


图 19-50 安装 JRockit

(3) 可以在如图 19-50 所示的窗口中设置是否安装 JRockit JRE，默认情况下不安装。在此选中“**Yes**”单选按钮（安装 JRockit JRE）。单击“**Next**”按钮继续安装。此时界面如图 19-51 所示。

提示 JRockit JRE 是 BEA 公司开发的 Java 运行时环境。

(4) 可以在如图 19-51 所示的窗口中设置使用 JRockit 插件的浏览器。在此使用默认设置。单击“**Next**”按钮继续安装。此时界面如图 19-52 所示。

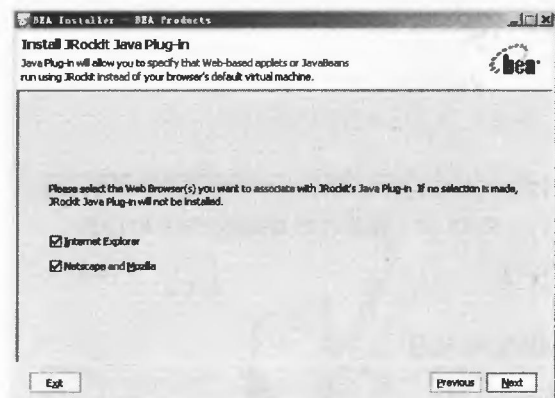


图 19-51 选择使用 JRockit 的 Java 插件的浏览器



图 19-52 安装界面

(5) 图 19-52 中显示了安装进度，根据机器配置的不同，该安装过程可能会持续不等的一段时间。安装完毕以后，界面如图 19-53 所示。

(6) 单击“**Done**”按钮，此时就已经成功的完成了 Weblogic 9.0 的安装。



图 19-53 安装完成界面

19.6.3 配置 Weblogic 9.0

安装完 Weblogic 9.0 以后，需要进行相应的配置才能使用。按照如下步骤完成 Weblogic 9.0 的配置。

(1) 单击“开始”按钮，依次选择“程序”/“BEA Products”/“Tools”/“Configuration Wizard”选项，界面如图 19-54 所示。

(2) 可以在如图 19-54 所示的窗口中选择创建新域（Create a new WebLogic domain）或者扩展已有域（Extend an existing WebLogic domain），在此选择创建新域。单击“Next”按钮继续配置，此时界面如图 19-55 所示。

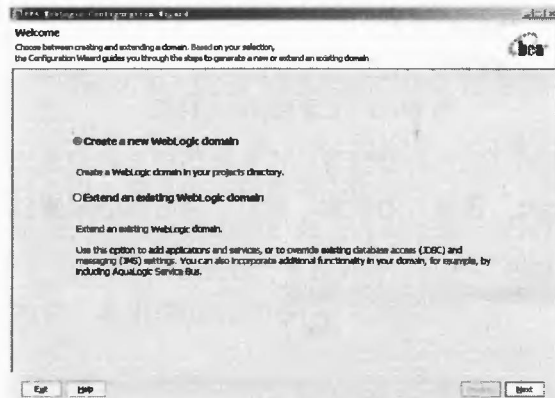


图 19-54 创建新域或者扩展已有的域

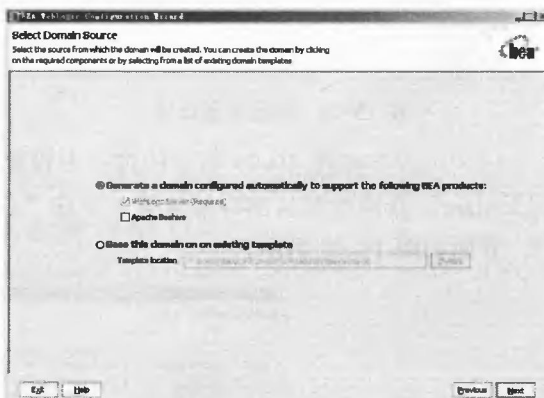


图 19-55 设置创建域所需要的资源

(3) 可以在如图 19-55 所示的窗口中设置创建域所使用的资源，在此各选项均使用默认值。单击“Next”按钮继续设置，此时界面如图 19-56 所示。

(4) 可以在图 19-56 中设置管理员账号和密码。设置完毕以后，单击“Next”按钮继续配置，此时窗口如图 19-57 所示。

(5) 可以在图 19-57 中设置启动当前域时的启动模式以及当前域使用的 JDK。在此均使用默认设置，单击“Next”按钮继续配置，此时窗口如图 19-58 所示。

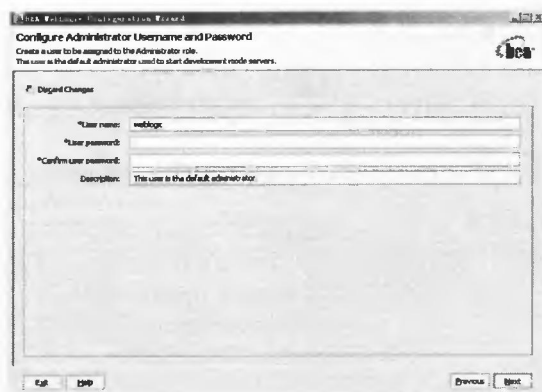


图 19-56 设置管理员账号和密码

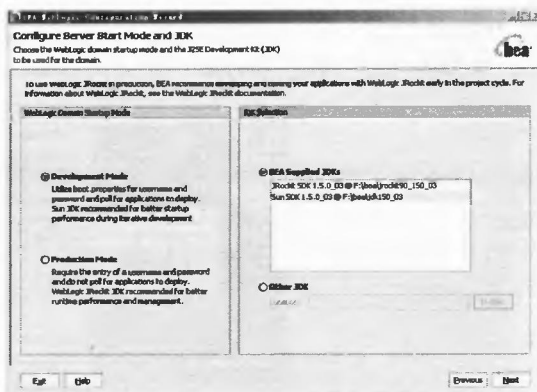


图 19-57 设置启动模式以及当前域使用的 JDK

(6) 图 19-58 中给出了其他一些可以设置的选项，如果要对这些选项进行设置，可以选定“**Yes**”单选按钮，如果暂时不需要设置这些选项，则选定“**No**”单选按钮。在此选中“**No**”单选按钮后，单击“**Next**”按钮继续配置，此时界面如图 19-59 所示。

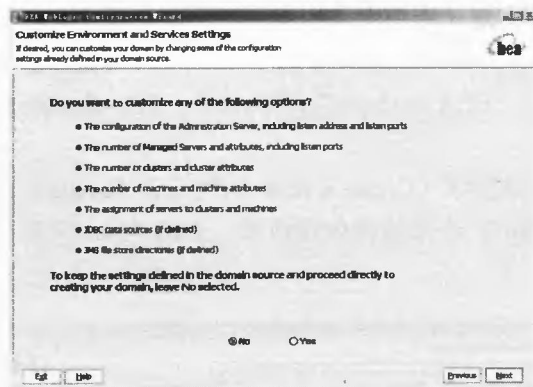


图 19-58 其他设置选项



图 19-59 设置域名称与位置

(7) 可以在如图 19-59 所示的窗口中设置域的名称与存储位置。在此将域的名称设置为 mydomain，并使用默认的存储位置。单击“**Create**”按钮开始配置。经过一段时间的配置以后，界面如图 19-60 所示。



图 19-60 配置信息界面

(8) 在如图 19-60 所示的窗口中显示了一些配置信息。单击“Done”按钮完成配置。

19.6.4 测试 Weblogic 9.0

前面几个小节介绍了 Weblogic 9.0 的安装与配置, 本节将对 Weblogic 9.0 进行测试, 以检验其是否成功地安装以及配置。按照如下步骤进行测试。

(1) 单击“开始”按钮, 依次选择“程序”/“BEA Products”/“Weblogic Server 9.0”选项。经过一段时间的启动工作以后, 界面如图 19-61 所示。

(2) 如果在窗口中出现了“start in RUNNING mode”信息, 表明 Weblogic 9.0 已经启动。此时打开浏览器, 在浏览器的地址栏内输入如下内容:

`http://localhost:7001/`

(3) 此时如果出现如图 19-62 所示的欢迎界面, 表明 Weblogic 9.0 安装成功。

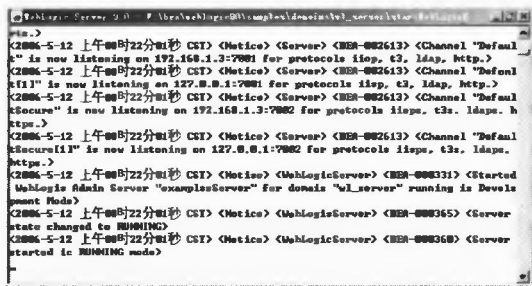


图 19-61 启动 Weblogic 9.0 窗口

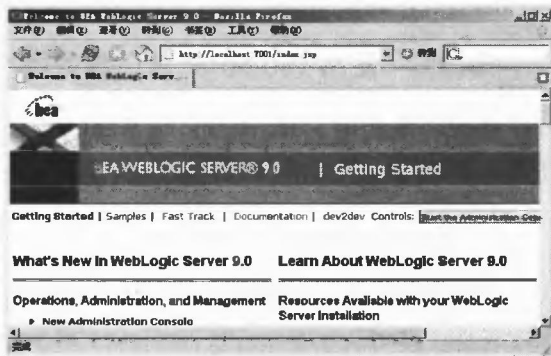


图 19-62 Weblogic 9.0 的欢迎页面

19.6.5 连接 Weblogic 9.0 服务器

连接 NetBeans 和 WebLogic 9.0 的步骤如下。

(1) 用鼠标右键单击“Runtime”窗口中的 Servers 节点, 选择右键菜单的“Add Server...”选项, 系统会弹出添加服务器实例的向导窗口, 如图 19-63 所示。

(2) 在“Server”下拉列表中选择“BEA WebLogic Application Server 9.0”, 单击“Next”按钮, 将出现如图 19-64 所示的窗口。

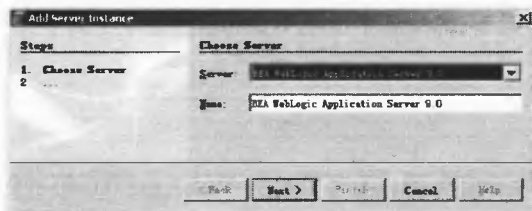


图 19-63 添加服务器实例的向导窗口

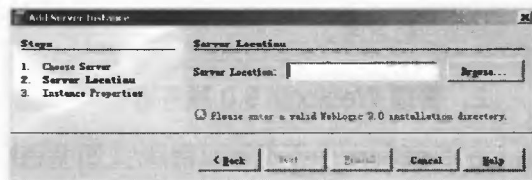


图 19-64 选择服务器安装位置的向导窗口

(3) 首先单击“Browse...”按钮选择安装 WebLogic 9.0 服务器的位置, 如: “F:\bea90\weblogic90”, 接着单击“Next”按钮进入服务器连接信息设置窗口, 如图 19-65 所示。

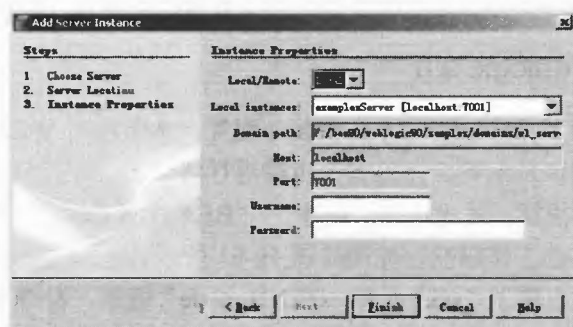


图 19-65 服务器连接信息设置窗口

(4) 在“Username”和“Password”字段中分别输入进入 WebLogic 服务器的用户名和密码，单击“Finish”按钮完成与服务器的连接。连接成功后“Servers”节点下会出现一个 **bea** 图标节点，这个节点代表与 WebLogic 服务器的连接。

19.6.6 使用 Weblogic 9.0 服务器

在成功地连接了 NetBeans 与 Weblogic 9.0 之后就可以对 Weblogic 9.0 服务器进行操作，主要包括启动、关闭和管理 Weblogic 9.0 服务器。

1. 启动 Weblogic 9.0 服务器

启动 Weblogic 9.0 服务器的方法非常简单，在 **bea** 图标节点上单击鼠标右键，选择右键菜单中的“Start”选项，这时 NetBeans 就会开始启动 Weblogic 9.0 服务器，同时在 NetBeans 中出现显示 Weblogic 9.0 服务器启动信息的窗口，如图 19-66 所示。

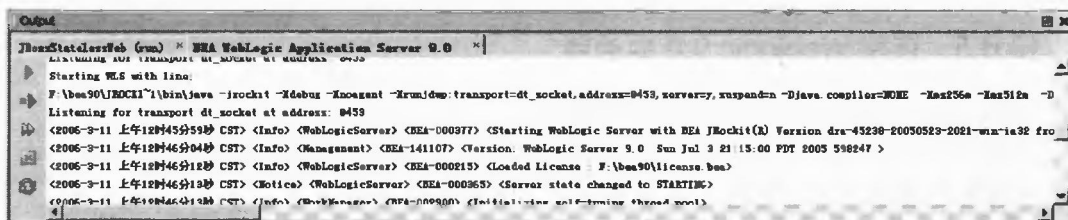


图 19-66 服务器启动信息的窗口

如果要关闭 Weblogic 9.0 服务器，则在 **bea** 图标节点上单击鼠标右键，选择右键菜单中的“Stop”选项即可，同时也可以看到 NetBeans 中出现 Weblogic 9.0 服务器关闭的信息。

2. 管理 Weblogic 9.0 服务器

在 NetBeans 中不但可以启动/关闭 Weblogic 9.0 服务器，还可以打开 Weblogic 9.0 服务器的管理页面。只要用鼠标右键单击 **bea** 图标节点，选择右键菜单中的“View Admin Console”选项即可，这时系统会自动启动默认浏览器，并指定浏览器的 URL 为 Weblogic 9.0 服务器的管理登录页面，如图 19-67 所示。

这时只要在浏览器中输入进入 WebLogic 的用户名和密码即可进入 Weblogic 9.0 服务器的

管理页面，读者可以自行操作，这里不再赘述。

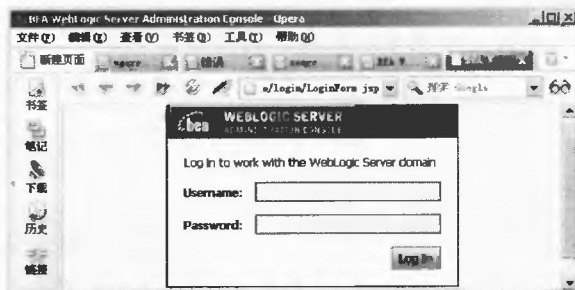


图 19-67 浏览器中 WebLogic 9.0 的管理登录页面



提示

只有在 WebLogic 服务器已经成功启动的情况下才可以对其进行管理，如果服务器没有正确启动则“View Admin Console”选项为灰色不可操作。

19.7 配置 JBoss4.0

前面已经介绍了如何在 NetBeans 中配置 WebLogic 9.0，下面将介绍如何在 NetBeans 中配置另一款目前主流的 Java EE 应用服务器—JBoss 4.0。

19.7.1 安装与测试 JBoss 4.0

JBoss 也是一款完全免费的 Java EE 应用服务器，可以从 www.jboss.org 下载其相应的版本。JBoss 不同于 WebLogic，下载以后的 JBoss 是一个 zip 格式的压缩文件。将其解压到任意一个目录后（如 F:\jboss4），打开 JBoss 4.0 目录中下的 bin 目录，双击 run.bat 文件即可以启动 jboss，界面如图 19-68 所示。

如果在图 19-68 所示窗口中出现“Started in XXX ms”，则说明 JBoss 已经启动完毕。此时打开浏览器，在浏览器的地址栏中输入如下内容：

`http://localhost:8080`

如果浏览器中出现如图 19-69 所示内容，则表明成功启动。

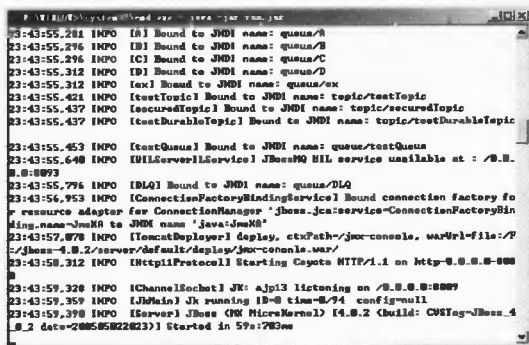


图 19-68 启动界面

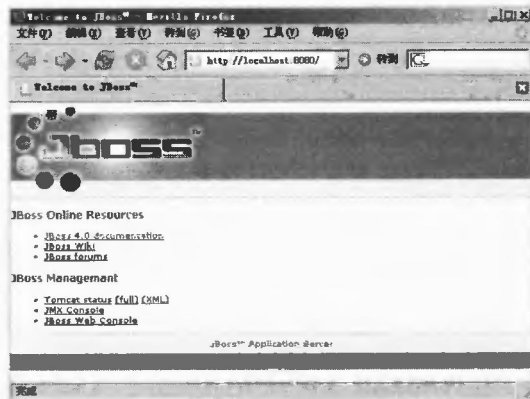


图 19-69 测试是否安装成功

如果不能成功启动 JBoss，可能是环境变量配置不正确所致。可以按照如下步骤进行环境变量的设置。

(1) 在“我的电脑”图标上单击鼠标右键，选择右键菜单中的“属性”选项。在弹出的窗口中选择“高级”选项卡，窗口如图 19-70 所示。

(2) 单击“环境变量”按钮，出现如图 19-71 所示的窗口。

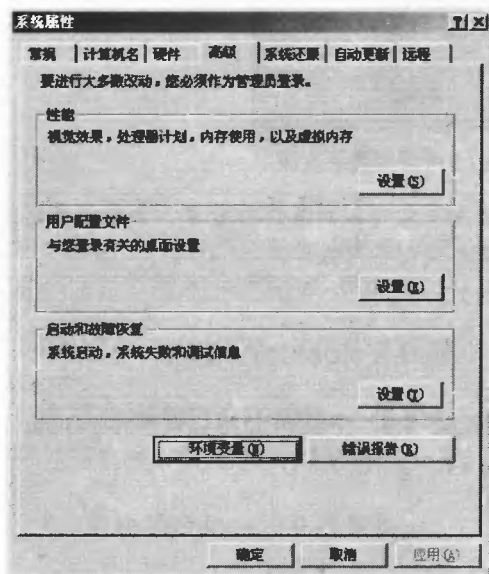


图 19-70 “高级”选项卡

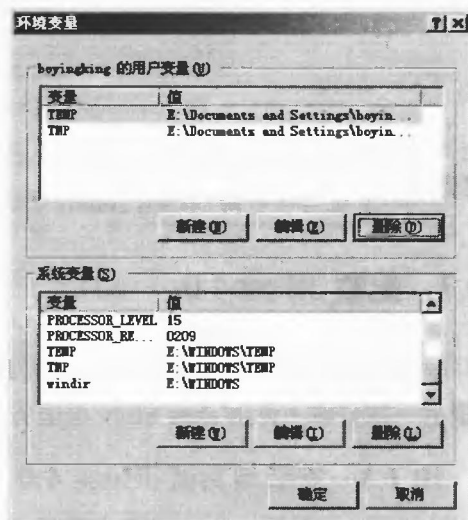


图 19-71 配置环境变量窗口

(3) 单击“系统变量”面板中的“新建”按钮，出现如图 19-72 所示的窗口。

(4) 可以在如图 19-72 所示的窗口中设置要添加的环境变量的名称和变量值，接着单击“确定”按钮即可完成添加。按如表 19-2 所示添加环境变量。

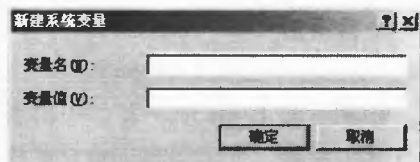


图 19-72 添加系统变量

表 19-2 添加的环境变量的变量名与值

变 量 名	变 量 值
JAVA_HOME	“F:\Java\jdk1.5.0_03”，其为 JDK 的安装路径
JBoss_HOME	“F:\jboss4”，其为 JBoss 的安装路径
PATH	F:\Java\jdk1.5.0_03\bin;%PATH%，其中“F:\Java\jdk1.5.0_03\bin”为 JDK 的 bin 目录

提示 环境变量的具体值要根据读者实际的安装情况确定。

此时就已经成功地完成了环境变量的设置。

19.7.2 在 NetBeans 中配置 JBoss 4.0

在 NetBeans 中配置 JBoss 4.0 的具体步骤如下。

(1) 用鼠标右键单击“Runtime”窗口中的 Servers 节点，选择右键菜单中的“Add Server...”

选项，系统会弹出添加服务器实例的向导窗口。

(2) 在“Server”下拉列表中选择“JBoss Application Server 4.0.3”，单击“Next”按钮。

(3) 在弹出的向导窗口单击“Browse...”按钮选择安装 JBoss 4.0 服务器的位置，如 F:\jboss4，接着单击“Next”按钮进入服务器连接信息设置窗口，如图 19-73 所示。

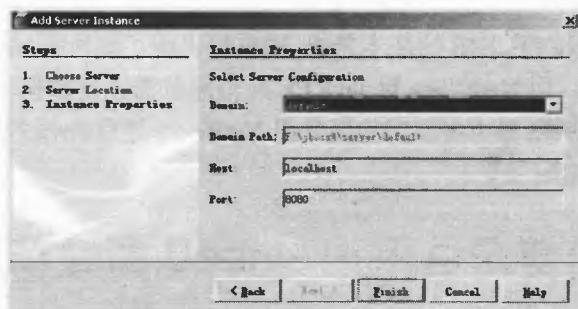



图 19-73 服务器连接信息设置窗口

(4) 在“Domain”下拉列表中选择要连接的域，单击“Finish”按钮就完成了 JBoss 4.0 与 NetBeans 的连接。连接成功后，在 Servers 节点下会出现一个  图标节点，这个节点代表与 JBoss 服务器的连接。

启动与关闭 JBoss 服务器的方法和 WebLogic 相同，读者可以参照 19.6.6 的内容自行操作。NetBeans 还可以连接其他的一些应用服务器，如 Sun 公司的 SJS 等，方法和 JBoss 以及 WebLogic 基本相同，读者也可以参照前面的章节自行完成。

19.8 在 NetBeans 中管理 DTD/XML 库

在用 NetBeans 进行 Java EE 开发的时候可能会用到很多 DTD 或 XML 文件库，比如 JSF (Java Server Faces)、Struts 的标签库等。在安装了 NetBeans 后，系统已经自带了很多的文件库，当展开运行时窗口中的 DTD and XML Schema Catalogs 节点时可以看到系统中已经安装的 DTD 和 XML 文件库。

提示 其中有的节点标识为[read-only]，有的标识为[read-write]，只有标识为[read-write]的文件库才可以进行修改。

如果开发过程中需要用到新的 DTD/XML 文件库，可以在 NetBeans 中进行添加，具体的步骤如下。

(1) 用鼠标右键单击“Runtime”窗口中的 DTD and XML Schema Catalogs 节点，选中右键菜单中的“Add Catalog...”选项，系统将弹出添加向导窗口，如图 19-74 所示。

(2) 在“Catalog Type”下拉列表中选择要添加的库的种类，单击“Browse”按钮选择要添加

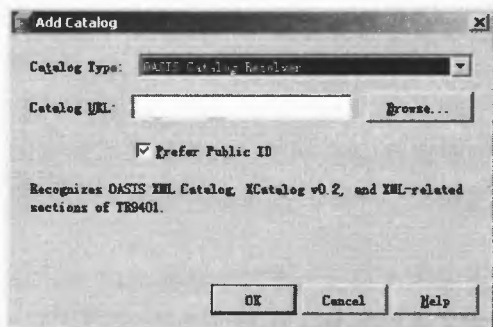


图 19-74 添加向导窗口

的库文件的路径,再单击“OK”按钮即可完成新库的添加。添加成功后可以在“DTD and XML Schema Catalogs”节点下看到代表新库的子节点。

19.9 小结

在本章中介绍了 Java EE、EJB 的基本思想,同时也讲述了如何在 NetBeans 中配置 Java EE 的开发环境。可以发现,用 NetBeans 对 Java EE 应用服务器以及数据库服务器进行管理是非常方便快捷的。在下面的章节中将介绍如何使用 NetBeans 进行 Java EE 应用的开发。

第 20 章

无状态会话 Bean——用户消费信息登记

在本章中将向读者介绍如何使用 NetBeans 进行无状态会话 Bean 的开发。首先介绍什么是无状态会话 Bean，接着通过一个具体的例子进行详细介绍。主要包括如下内容：

- 无状态会话 Bean 简介；
- 项目简介；
- 建立数据库表；
- 创建项目；
- 开发 EJB 模块；
- 开发 Web 模块；
- 编译运行整个项目。

20.1 无状态会话 Bean 简介

无状态会话 Bean 主要用来实现单次使用的服务，该服务能被启用许多次，但是由于无状态会话 Bean 并不保留任何有关状态的信息，其效果是每次调用提供单独的使用。在很多情况下，无状态会话 Bean 提供可重用的单次使用服务。

尽管无状态会话 Bean 并不为特定的客户维持会话状态，但会有一个以其成员变量形式表示的过度状态。当一个客户调用无状态会话 Bean 的方法时，Bean 的成员变量的值只表示调用期间的一个过度状态。当该方法完成时，这个状态不再保留。

除了在方法调用期间，所有的无状态会话 Bean 的实例都是相同的，允许 EJB 容器给任何一个客户赋予一个实例。许多应用服务器利用这个特点，共享无状态会话 Bean 以获得更好的性能。

由于无状态会话 Bean 能够支持多个客户，并且通常在 EJB 容器中共享，可以为需要大量客户的应用提供更好的扩充能力。无状态会话 Bean 比有状态会话 Bean 更具优势的是其性能，在条件允许的情况下开发人员应该首先考虑使用无状态会话 Bean。

20.2 项目简介

上节对无状态会话 Bean 进行了简单的介绍，本节将对本章中开发的项目进行介绍，主要包括如下内容：

- 项目功能介绍；
- 模块结构介绍。

20.2.1 项目功能演示

本节将对该项目的功能进行演示，步骤如下。

(1) 运行项目，出现用户消费信息的输入界面，如图 20-1 所示。

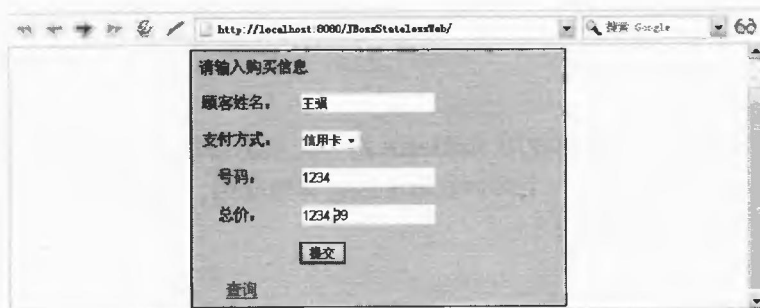


图 20-1 消费信息输入界面

(2) 在浏览器界面中用户可以输入姓名、支付方式、信用卡或支票号以及消费的总价格等信息，信息输入完毕后，单击“提交”按钮，数据将被写进数据库，写入成功后，浏览器会跳转到如图 20-2 所示的页面。

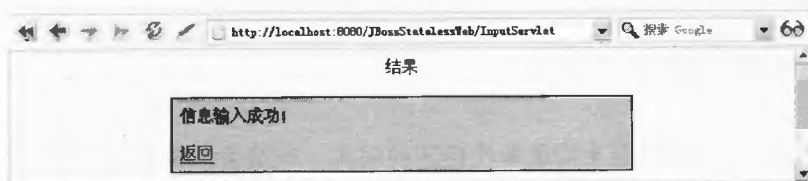


图 20-2 信息输入成功提示页面

(3) 如果写入失败，也就是输入的信息有错误，则浏览器会显示信息输入失败的页面，如图 20-3 所示。

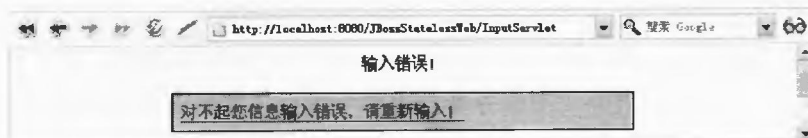


图 20-3 信息输入失败页面

(4) 在如图 20-2 所示的页面中单击返回超级链接可以返回消费信息输入界面。如果在消费信息输入界面中单击查询超级链接可以进入消费信息查询页面，如图 20-4 所示。

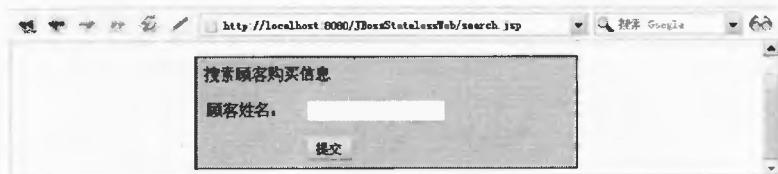


图 20-4 消费信息查询页面

(5) 在此页面中用户可以输入需要查询的客户姓名，单击“提交”按钮，浏览器将显示查询结果页面。如果查询成功，浏览器中将显示用户的消费信息。如输入“王强”进行查询，结果如图 20-5 所示。

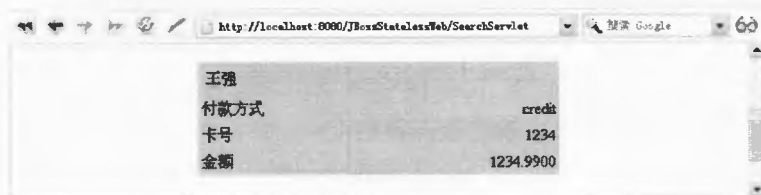


图 20-5 查询结果页面

(6) 如果查询的信息不存在，则显示如图 20-6 所示的页面。

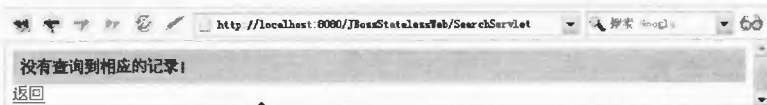


图 20-6 查询信息不存在页面

20.2.2 模块结构介绍

本项目是使用 NetBeans 结合 JBoss 开发一个无状态会话 Bean，并通过 Web 应用调用这个无状态会话 Bean。在本例中无状态会话 Bean 负责的功能，主要是向数据库中添加用户进行消费的数据，以及在数据库中进行查询。

在本项目中有一个无状态会话 Bean——“PaymentBean”，其负责用户与数据库的交互。另外还有 5 个 JSP 页面负责表示层、两个 Servlet 负责控制器的功能，这些组件的功能如表 20-1 所示。

表 20-1 各个 JSP 页面的功能

页 面	功 能
input.jsp	负责担任用户消费信息输入的界面
result.jsp	负责显示用户信息输入成功提示界面
error.jsp	负责显示用户信息输入失败提示界面
search.jsp	负责显示查询用户消费信息的界面
searchresult.jsp	负责显示查询结果的界面
InputServlet	负责处理用户的输入请求
SearchServlet	负责处理用户的查询请求

项目中所有 JSP、Servlet 以及 EJB 的功能关系如图 20-7 所示。

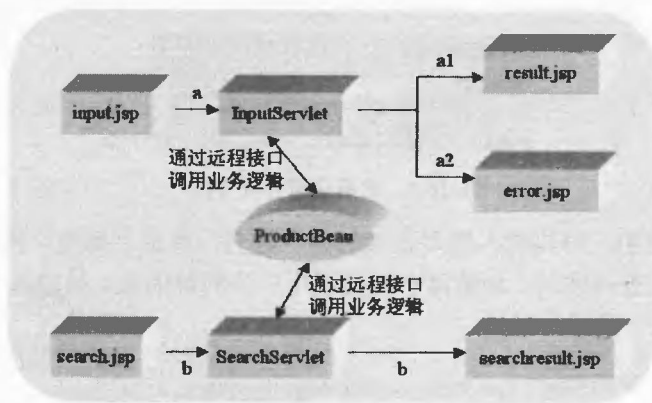


图 20-7 项目的逻辑结构图

表 20-2 给出了各个编号所对应的动作内容。

表 20-2 动作编号对照表

动作编号	动作内容	动作编号	动作内容
a	提交用户输入信息	a1	用户信息输入成功
a2	用户输入信息失败	b	查询商品

20.3 建立数据库表

由于在本项目中要使用数据库，因此在正式开始编写程序之前，要在数据库中创建相应的数据库表。在这里选用开源数据库 MySQL 作为项目的后台数据库，在数据库中创建一个名为 payment_table 的表，表中各个字段的名称和类型如表 20-3 所示。具体的创建过程读者可以参照上一章，这里不再赘述。

表 20-3 表 payment_table 中的各个字段

字段名称	字段类型	约束条件
Customer	varchar(20)	NOT NULL
Payment_type	varchar(20)	NOT NULL
Card_number	varchar(20)	
Amount	NUMERIC(10,4)	NOT NULL

在 EJB 中访问数据库，一般不会直接编写 JDBC 的代码，而是使用数据源和连接池。在成功地创建了数据库表后，就要在 JBoss 中配置相应的数据源。

JBoss 是一个开源的应用服务器，它不像 WebLogic 这样的商业应用服务器有友好的用户配置界面，在 JBoss 中配置数据源的方式就是编写相应的 XML 配置文件。在本项目中编写一个名为 mysql-ds.xml 的 XML 文件，并把该文件存放到 JBoss 安装目录的 server\default\deploy 中。mysql-ds.xml 配置文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
```

```

<local-tx-datasource>
  <jndi-name>wzthhb</jndi-name>
  <connection-url>jdbc:mysql://localhost:3306/test</connection-url>
  <driver-class>org.gjt.mm.mysql.Driver</driver-class>
  <user-name>root</user-name>
  <password></password>
  <min-pool-size>5</min-pool-size>
  <max-pool-size>20</max-pool-size>
  <idle-timeout-minutes>0</idle-timeout-minutes>
  <track-statements/>
</local-tx-datasource>
</datasources>

```

- <jndi-name>标签中包含的是数据源的 JNDI 名称。
- <connection-url>标签中包含的是连接 MySQL 数据库的连接字符串。
- <driver-class>标签指出连接所使用的驱动程序。
- <user-name>标签指出要连接的数据库的用户名。
- <password>标签指出要连接的数据库的密码，如果没有密码则为空。
- <min-pool-size>指出连接池的最少连接数。
- <max-pool-size>指出连接池的最大连接数。

在编写完配置文件后，在 NetBeans 中启动 JBoss 服务器，测试数据源是否创建成功。如果数据源配置正确，在启动信息窗口中，就会看到数据源创建成功的信息，如图 20-8 所示。

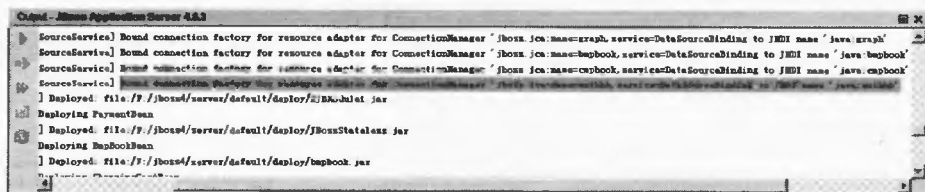


图 20-8 数据源创建成功

20.4 创建项目

从本节开始，将详细介绍如何使用 NetBeans 开发无状态会话 Bean，创建项目的具体步骤如下。

(1) 启动 NetBeans，依次选择主菜单中的“File”/“New Project...”选项，在弹出的新建项目窗口中选择项目分类“Enterprise”中的“Enterprise Application”项目类型，如图 20-9 所示。

(2) 单击“Next”按钮，进入项目信息设置窗口，如图 20-10 所示。

(3) 设置项目的名称为“StatelessBeanExample”，在“Server”下拉列表中选择“JBoss Application Server 4.0.3”，选中“Create EJB Module”和“Create Web Application Module”选项。

(4) 单击“Finish”按钮完成项目的创建。在项目创建完成后会发现在“Project”窗口中

出现了以▲图标代表的节点，此节点表示企业级应用项目。展开该节点，会发现下面有两个子节点，其中用●图标表示的是 EJB 项目模块，另一个是 Web 应用模块，如图 20-11 所示。

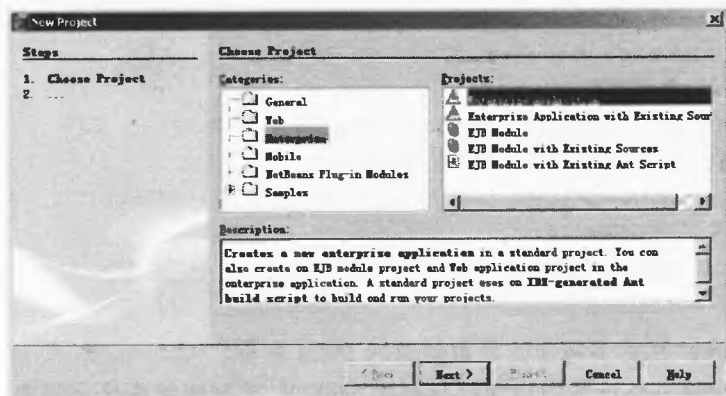


图 20-9 新建项目窗口

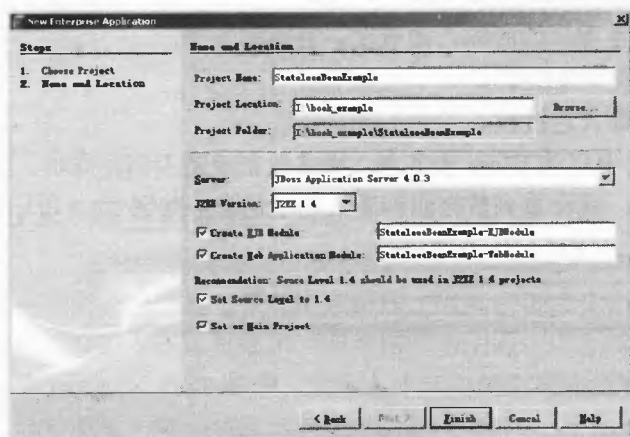


图 20-10 项目信息设置窗口

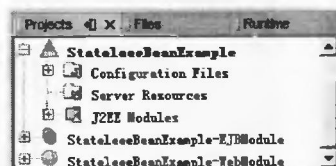


图 20-11 创建完成后的项目窗口

20.5 开发 EJB 模块

在本节中将介绍本例中 EJB 模块的开发，开发过程主要包括使用 NetBeans 快速开发代码框架以及向代码框架中加入业务逻辑代码。

20.5.1 开发 EJB 代码框架

在本小节中将介绍如何使用 NetBeans 可视化快速开发无状态会话 Bean 代码框架，步骤如下。

(1) 展开在上一节中创建项目的 EJB 模块节点 (StatelessBeanExample-EJBModule 节点)，选中 Enterprise Beans 子节点，如图 20-12 所示。

(2) 单击鼠标右键，在弹出的菜单中依次选择“New”/“Session Bean...”选项，系统将弹出新建会话 Bean 向导窗口，如图 20-13 所示。



图 20-12 项目窗口中的 EJB 模块

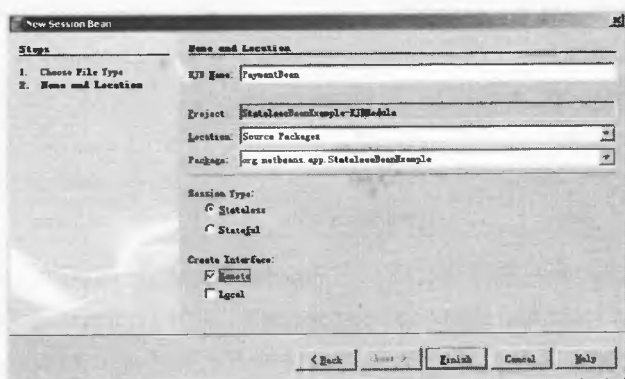


图 20-13 新建会话 Bean 向导窗口

(3) 设置 EJB 的名称为“PaymentBean”，指定其所在的包名为“org.netbeans.app.StatelessBeanExample”，选择会话 Bean 的类型为“Stateless”，选中“Remote”选项。

(4) 单击“Finish”按钮，完成无状态会话 Bean 的创建。创建成功后在项目窗口中 Enterprise Beans 节点下将出现 PaymentSB 节点，如图 20-14 所示。

(5) 用鼠标右键单击 PaymentSB 节点下 Remote Methods 子节点，依次选择右键菜单中的“Add”/“Business Method...”选项，系统会弹出创建业务方法向导窗口，如图 20-15 所示。

(6) 指定新建的业务方法的名称为“payByCash”，返回值类型为“boolean”。单击“Add...”按钮为方法添加入口参数，这时系统会弹出添加参数向导窗口，如图 20-16 所示。

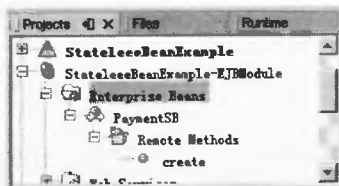


图 20-14 PaymentSB 节点

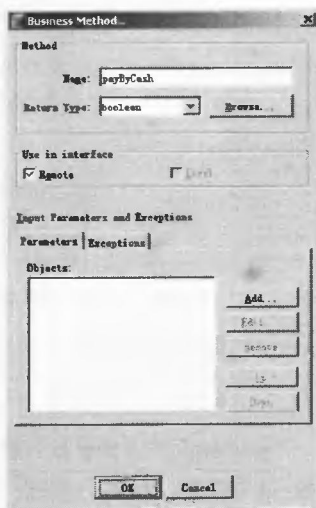


图 20-15 创建业务方法向导窗口

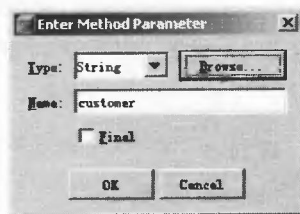


图 20-16 添加参数向导窗口

(7) 指定参数的类型为“String”，名称为“customer”，单击“OK”按钮完成参数的添加。

(8) 用同样的方法再添加一个名称为“amount”，类型为“double”的参数。参数添加成后你会发现创建业务方法向导窗口中的“Parameters”列表中多出了两个参数，如图 20-17 所示。

(9) 单击“OK”按钮，完成“payByCash”业务方法的创建。创建成功后会发现“Project”窗口中 Remote Methods 节点下出现了以图标表示的 payByCash 节点，即新创建的业务方法。

(10) 按照上面的步骤再添加 3 个业务方法, 详细信息如表 20-4 所列。

表 20-4 3 个业务方法的详细信息

方法名称	返回值类型	参数序列
payByCheck	boolean	String customer, String check_number, double amount
payByCreditCard	boolean	String customer, String card_number, double amount
search	java.util.Vector	String customer

(11) 依次展开项目窗口中 StatelessBeanExample-EJBModule 节点下的“Source Packages”/“org.netbeans.app.StatelessBeanExample”/“PaymentBean.java”/“PaymentBean”/“Methods”节点, 单击鼠标右键, 在弹出的菜单中选择“Add Method...”选项, 系统会弹出添加方法向导窗口, 如图 20-18 所示。

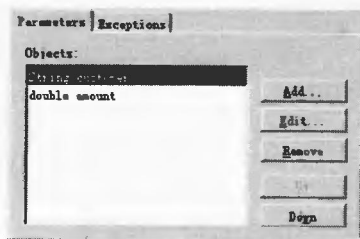


图 20-17 新添加的参数

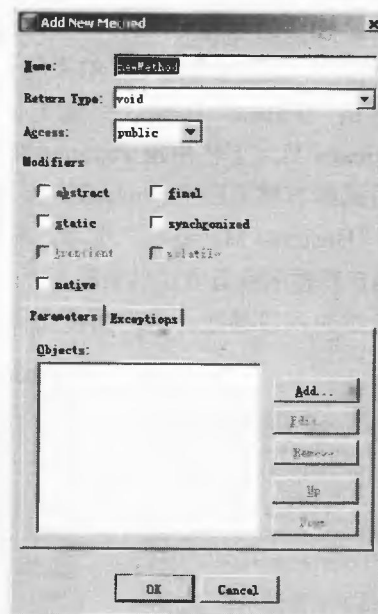


图 20-18 添加方法向导窗口

(12) 设置方法的名称为“process”, 返回值类型为“boolean”, 访问限制修饰符为“private”, 同时给方法添加 4 个参数, 参数的类型名称如表 20-5 所示。

表 20-5 “process”方法的参数列表

参数名称	参数类型	参数名称	参数类型
cusName	String	cardNumber	String
payType	String	amt	double

(13) 单击“OK”按钮完成方法的添加, 方法添加成功后会发现在 Methods 节点下出现了一个 process 节点, 这就是新添加的方法。

20.5.2 业务代码的开发

在本小节中将介绍本例中无状态会话 Bean 业务方法代码的开发, 步骤如下。

(1) 双击 `process` 节点, 这时光标会出现在代码编辑器中的 `process` 方法上, 在 `process` 方法体中添加如下代码:

```
boolean retType = false;
try{ java.sql.Connection conn = getConnection();
    java.sql.PreparedStatement ps = conn.prepareStatement("insert into
Payment_table (Customer_Name, Payment_type, Card_number, Amount) values
(?, ?, ?, ?)");
    ps.setString(1, cusName);
    ps.setString(2, payType);
    ps.setString(3, cardNumber);
    ps.setDouble(4, amt);
    int value = ps.executeUpdate();
    if(value == 1) retType = true;
    ps.close();
    conn.close();
}catch(Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
return retType;
```

- 这个方法的功能是把收到的信息用 `PreparedStatement` 写入数据库。
- 如果写入成功返回值为 “true”, 否则为 “false”。

(2) 用同样的方法向 `PaymentBean` 节点下 `Methods` 节点中添加一个 `Connection` 方法, 该方法的签名为 “`private Connection getConnection() throws java.sql.SQLException`”。

(3) 在 `Connection` 方法的方法体中加入如下代码:

```
javax.naming.Context ctx = null;
javax.sql.DataSource ds = null;
String url = "localhost:1099";
try {java.util.Hashtable h = new java.util.Hashtable( );
    h.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
        "org.jnp.interfaces.NamingContextFactory");
    h.put(javax.naming.Context.PROVIDER_URL, url);
    ctx = new javax.naming.InitialContext(h);
    ds = (javax.sql.DataSource) ctx.lookup("java:wzthhb");
} catch(Exception ne) {
    System.out.println("UNABLE to get a connection from !");
}
return ds.getConnection();
```

- 这个方法的作用是通过 `JNDI` 技术获取数据库链接。
- 如果成功获取链接则返回连接对象的句柄, 否则返回 “null”。

(4) 用鼠标双击 `payByCash` 节点, 这时光标会出现在代码编辑器中的 `payByCash` 方法上, 在 `payByCash` 方法体中加上如下代码:

```
return process(customer, "cash", " ", amount);
```

提示 实际上 `payByCash` 方法自己并不处理数据，而是把数据传给 `process` 方法处理。

(5) 在 “`payByCheck`” 方法和 “`payByCreditCard`” 方法中分别添加如下两行代码：

```
return process(customer, "check", check_number, amount); // 添加到 payByCheck 方法中
```

```
return process(customer, "credit", card_number, amount); // 添加到 payByCreditCard 方法中
```

(6) 在 “`search`” 方法的方法体中添加如下代码：

```
java.util.Vector vec = new java.util.Vector();
try{ java.sql.Connection conn = getConnection();
    java.sql.PreparedStatement ps = conn.prepareStatement("select *
from Payment_table where Customer_Name = ?");
    ps.setString(1, customer);
    java.sql.ResultSet rs = ps.executeQuery();
    while (rs.next()) {
        String temp[] = new String[4];
        temp[0]=rs.getString("customer_name");
        temp[1]=rs.getString("payment_type");
        temp[2]=rs.getString("card_number");
        temp[3]=rs.getString("amount");
        vec.add(temp);
        temp = null;
    }
    rs.close();
    ps.close();
    conn.close();
}catch(Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
return vec;
```

- 这个方法的主要功能是根据输入的用户名查找相关用户的消费信息，并把信息封装在 `Vector` 对象中返回。
- 在表示层的 JSP 中可以把 `Vector` 对象中的内容显示出来。

20.5.3 配置 EJB 模块

经过前面两个小节的工作，EJB 模块的开发工作已经完成。本小节将对该 EJB 模块进行配置，按照如下步骤完成。

(1) 双击项目窗口中 “`StatelessBeanExample-EJBModule`” / “`Configuration Files`” / `jboss.xml` 节点，在代码编辑框中把 `jboss.xml` 文件的内容替换为如下内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC
```

```

"-//JBoss//DTD JBOSS 4.0//EN"
"http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>PaymentBean</ejb-name>
      <jndi-name>PaymentBeanNetBeans</jndi-name>
    </session>
  </enterprise-beans>
</jboss>

```

- <ejb-name>标签指出 EJB 的名称，此名称要和前面一致。
- <jndi-name>指出 EJB 的 JNDI 名称。

(2) 用鼠标右键单击项目窗口中 StatelessBeanExample-EJBModule 节点，选择右键菜单的“Clean and Build Project”选项编译并打包 EJB 模块。如果编译打包成功，在输出窗口中将显示成功的信息，如图 20-19 所示。

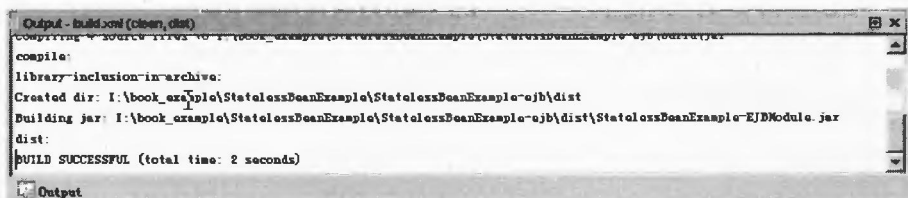


图 20-19 输出编译打包成功信息的输出窗口

20.6 开发 Web 模块

在上一节中已经开发了负责业务逻辑的 EJB 模块，在本小节中将介绍如何开发用于和终端用户交互的 Web 模块。Web 模块中的组成部分已经在表 20-1 中给出，下面将详细介绍 Web 模块的开发过程。

20.6.1 开发用户输入页面与信息输入成功页面

在本小节中将介绍本例中输入页面与查询结果页面开发，具体步骤如下。

(1) 展开项目窗口中的“StatelessBeanExample-WebModule”节点，用鼠标右键单击 Web Pages 子节点，依次选择右键菜单中的“New”/“JSP...”选项，系统会弹出新建 JSP 文件向导窗口。

(2) 在“JSP File Name”字段中指定 JSP 文件的名称为“input”，单击“Finish”按钮完成 JSP 文件的创建。文件创建成功后，会在 Web Pages 节点下出现一个 input.jsp 子节点，这就是新创建的 JSP 文件。

(3) 双击 input.jsp 节点，这时光标会出现在代码编辑窗口中，把 input.jsp 文件的内容替换为如下代码：

```

<%@ page contentType="text/html; charset=GBK"%>
<html>

```

```

<head> <title>input</title> </head>
<body>
  <form name="form1" method="post" action="<%=request.getContextPath()%>/
InputServlet">
    <p align="center">&nbsp;</p>
    <table width="52%" border="0" align="center" cellpadding="1" cellspacing="
0" bgcolor="#000000">
      <tr>
        <td> <table width="100%" height="100%" border="0" cellpadding="8"
cellspacing="0" bgcolor="#CCCCCC">
          <tr>
            <td colspan="2" bgcolor="#CCCCCC">
              <font color="#000000">请输入购买信息</font>
            </td>
          </tr>
          <tr>
            <td width="27%" align="center">顾客姓名: </td>
            <td width="73%" align="left"><input type="text" name="custome-
rName"></td>
          </tr>
          <tr>
            <td align="center">支付方式: </td>
            <td width="73%" align="left">
              <select name="paymentType" size="1">
                <option value="cash" selected>现金</option>
                <option value="check">支票</option>
                <option value="credit">信用卡</option>
              </select></td>
            </tr>
            <tr>
              <td align="center">号码: </td>
              <td width="73%" align="left"> <input type="text" name="cardNu-
mber"> </td>
            </tr>
            <tr>
              <td align="center">总价: </td>
              <td width="73%" align="left"> <input type="text" name="cartVal-
ues"> </td>
            </tr>
            <tr>
              <td align="center">&nbsp;</td>
              <td width="73%" align="left">
                <input type="submit" name="Submit" value="提交">
              </td>
            </tr>
          <tr>

```

```

        <td align="center">
            <a href="<%=request.getContextPath()%>/search.jsp">查询</a>
        </td>
        <td width="73%" align="left">
        </td>
    </tr>
</table></td>
</tr>
</table>
</form>
</body>
</html>

```

- 这个 JSP 页的功能是把用户输入的消费信息，如姓名、支付方式、号码、总价等打包到一个 HTTP 请求中，发送给 InputServlet。
 - 第 5 行代码<%=request.getContextPath()%>的作用是获取 InputServlet 所在的路径。
- (4) 把“input.jsp”文件设置为本 Web 模块的欢迎页。
- (5) 按照上面的步骤在 Web 模块中添加 result.jsp 文件，result.jsp 文件代码如下：

```

<%@ page contentType="text/html; charset=GBK"%>
<html>
<head> <title></title> </head>
<body>
<p align="center">结果</p>
<table width="60%" border="0" align="center" cellpadding="1" cellspacing="0" bgcolor="#000000">
    <tr>
        <td> <table width="100%" border="0" cellpadding="8" cellspacing="0" bgcolor="#CCCCCC">
            <tr>
                <td colspan="2" bgcolor="#CCCCCC">
                    <font color="#000000">信息输入成功! </font>
                </td>
            </tr>
            <tr>
                <td colspan="2" bgcolor="#CCCCCC">
                    <font color="#000000">
                        <a href="<%=request.getContextPath()%>/input.jsp">返回</a>
                    </font>
                </td>
            </tr>
        </table></td>
    </tr>
</table>
<p>&nbsp;</p><p>&nbsp;</p>
</body>
</html>

```


● 用户在“input.jsp”中输入了相应的信息，并把信息提交给 InputServlet。

● 如果用户输入信息成功，则出现“信息输入成功!”的提示信息。

完成了上述步骤，就已经成功地完成了用户输入页面与信息输入成功页面的开发。

20.6.2 开发信息输入错误页面、查询页面

上个小节中完成了用户输入页面与信息输入成功页面的开发，本节将开发错误页面、查询页面以及查询结果页面，按照如下步骤完成这些页面的开发。

(1) 向 StatelessBeanExample-WebModule 中添加一个名为 error 的 JSP 文件。

(2) 删除 error.jsp 内自动生成的代码，并将下列代码添加到其中：

```
<%@ page contentType="text/html; charset=GBK"%>
<html>
<head><title></title></head>
<body>
<p align="center">输入错误! </p>
<table width="60%" border="0" align="center" cellpadding="1" cellspacing="0" bgcolor= "#000000">
  <tr>
    <td> <table width="100%" border="0" cellpadding="8" cellspacing="0" bgcolor="#CCCCCC">
      <tr>
        <td colspan="2" bgcolor="#CCCCCC">
          <font color="#000000">
            <a href="input.jsp">对不起您的信息输入错误，请重新输入! </a>
          </font>
        </td>
      </tr>
    </table></td>
  </tr>
</table>
<p>&nbsp;</p><p>&nbsp;</p>
</body>
</html>
```

(3) 向 StatelessBeanExample-WebModule 中添加一个名为 search 的 JSP 文件。

(4) 删除 search.jsp 内自动生成的代码，并将下列代码添加到其中：

```
<%@ page contentType="text/html; charset=GBK"%>
<html>
<head><title>search input page</title></head>
<body>
  <form name="form1" method="post" action="<%=request.getContextPath()%>/SearchServlet">
    <p align="center">&nbsp;</p>
    <table width="52%" border="0" align="center" cellpadding="1" cellspacing="0" bgcolor="#000000">
```



```

<tr>
  <td><table width="100%" height="100%" border="0" cellpadding="8"
    cellspacing="0" bgcolor="#CCCCCC">
      <tr>
        <td colspan="2" bgcolor="#CCCCCC">
          <font color="#000000">搜索顾客购买信息</font>
        </td>
      </tr>
      <tr>
        <td width="27%" align="center">顾客姓名: </td>
        <td width="73%" align="left"><input type="text" name="customerName"></td>
      </tr>
      <tr>
        <td align="center">&nbsp;</td>
        <td width="73%" align="left"><input type="submit" name="Submit"
          value="提交">
        </td>
      </tr>
    </table></td>
  </tr>
</table>
</form>
</body>
</html>

```

● “search.jsp”页面的作用是接收用户输入要查询的客户名称。

● 把客户名称提交给“SearchServlet”进行查询。

完成了上述步骤，就已经成功地完成了错误页面、查询页面以及查询结果页面的开发。

20.6.3 开发查询结果页面

本节将开发查询结果页面，按照如下步骤完成这些页面的开发。

- (1) 向 StatelessBeanExample-WebModule 中添加一个名称为 searchresult 的 JSP 文件。
- (2) 删除 searchresult.jsp 内自动生成的代码，并将下列代码添加到其中：

```

<%@ page import="java.util.*"%>
<%@ page contentType="text/html; charset=GBK"%>
<html>
<head><title>search result page </title></head>
<body>
<% Vector result = (Vector)request.getAttribute("result");
   if (result.size() == 0){
%>
<table width="100%" border="0" cellspacing="0" cellpadding="4">
  <tr>
    <td bgcolor="#CCCCCC" colspan="2"> <table width="100%" border="0"

```

```

cellspacing="0" cellpadding="4">
    <tr>
        <tdwidth="100%"><font color="#000000">没有查询到相应的记录! </font></td>
    </tr>
</table></td>
</tr>
</table>
<%>else{
String a[] = new String[4];
    for (Enumeration e = result.elements() ; e.hasMoreElements() ;) {
        a = (String[])e.nextElement();
    }
    <table width="50%" border="0" cellspacing="0" cellpadding="4" align="center">
        <tr>
            <td bgcolor="#CCCCCC" colspan="2">
                <table width="100%" border="0" cellspacing="0" cellpadding="4">
                    <tr>
                        <td width="100%"> <font color="#000000">
                            <%=new String(a[0].getBytes("ISO-8859-1"),"GB2312")%>
                        </font></td>
                    </tr>
                </table></td>
            </tr>
            <tr bgcolor="#CCCCCC">
                <td>付款方式</td> <td align="right"><%=a[1]%></td>
            </tr>
            <tr bgcolor="#CCCCCC">
                <td>卡号</td> <td align="right"><%=a[2]%></td>
            </tr>
            <tr bgcolor="#CCCCCC">
                <td>金额</td> <td align="right"><%=a[3]%></td>
            </tr>
        </table>
        <p>&nbsp;</p>
    <%%>%>
    <a href="<%=request.getContextPath()%>/search.jsp">返回</a></body>
</html>

```

- searchresult.jsp 页面的功能是把查询的结果通过“SearchServlet”调用出来。
 - 本页面接收由“SearchServlet”传递的 Vector 对象（查询的结果打包在 Vector 对象中），并遍历 Vector 对象，显示所有相关的客户信息。
- 完成了上述步骤，就已经成功地完成了查询结果页面的开发。

20.6.4 控制器 Servlet 的开发

在本小节中将介绍本例中两个 Servlet 的开发，具体步骤如下。

(1) 依次选中项目窗口中“StatelessBeanExample-WebModule”/“Source Packages”节点, 单击鼠标右键, 在弹出的菜单中依次选择“New”/“Servlet...”选项, 系统会弹出新建 Servlet 向导窗口。

(2) 在“Class Name”字段中指定 Servlet 类的名称为“InputServlet”, 单击“Next”按钮进入 Servlet 部署配制窗口。

(3) 在“URL Patterns”字段中指定 Servlet 的路径为/InputServlet, 单击“Finish”按钮完成 Servlet 的创建。创建成功后项目窗口中会出现 InputServlet.java 节点, 如图 20-20 所示。

(4) 用方法添加向导给 InputServlet 添加一个名称为“lookupHome”的方法, 方法的签名为“private org.netbeans.app.StatelessBeanExample.PaymentRemoteHome lookupHome()”。



图 20-20 “InputServlet.java”节点

(5) 在方法体中添加如下代码:

```
org.netbeans.app.StatelessBeanExample.PaymentRemoteHome myHome = null;
try { Hashtable ht = new Hashtable();
    ht.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
    ht.put(Context.PROVIDER_URL, "localhost:1099");
    Context ctx = new InitialContext(ht);
    Object home = ctx.lookup("PaymentBeanNetBeans");
    myHome = (PaymentRemoteHome) PortableRemoteObject.narrow(home,
        PaymentRemoteHome.class); //把远程对象引用转换成 Home 句柄
    } catch (Exception e) {
        System.out.println("The client was unable to lookup the
EJBHome."); } return myHome;
```

- 这个方法的主要功能是获取“PaymentBean”EJB 的 Home 句柄。
- ctx.lookup("PaymentBeanNetBeans")通过前面在 jboss.xml 文件中配置的 EJB JNDI 名称来获取远程对象引用。

(6) 再用方法添加向导给 InputServlet 添加一个名称为“forward”的方法, 方法的签名为“protected void forward (HttpServletRequest request, HttpServletResponse response, String url)” (此方法的功能是把请求进行转发)。

(7) 在方法体中添加如下代码:

```
RequestDispatcher dispatcher = request.getRequestDispatcher(url);
dispatcher.forward(request, response);
```

(8) 在“InputServlet”的“processRequest”方法中添加如下代码:

```
request.setCharacterEncoding("GBK");
boolean pay=false;
String paymentType = (String) request.getParameter("paymentType");
String cusname = (String) request.getParameter("customerName");
cusname=new String(cusname.getBytes(),"ISO-8859-1");
```

```

        String cardNumber = (String)request.getParameter("cardNumber");
        try { double cartValue = Double.parseDouble(request.getParameter("cartValues"));
            home = lookupHome();
            spayment = home.create();
            if(paymentType.equals("cash")) {
                pay = spayment.payByCash(cusname, cartValue);
            } else if (paymentType.equals("check")) {
                pay = spayment.payByCheck(cusname, cardNumber, cartValue);
            } else {
                pay = spayment.payByCreditCard(cusname, cardNumber, cartValue);
            }
        }
    } catch (Exception ce) { ce.printStackTrace(); }
    if (pay) {
        forward (request, response, "/result.jsp"); //输入成功后转到 //result.jsp 页
    } else {
        forward (request, response, "/error.jsp"); //输入失败后转到 //error.jsp 页
    }
}

```

- 此方法的功能是处理请求，用来被 doGet 和 doPost 方法调用。
- home = lookupHome(); 用来获取 EJB 的 Home 句柄。
- spayment = home.create(); 用来创建 EJB 对象。

(9) 在 “InputServlet” 中添加如下包的引用：

```

import java.util.*;
import javax.naming.*;
import javax.rmi.*;
import javax.ejb.*;
import org.netbeans.app.StatelessBeanExample.*;

```

(10) 用字段添加向导给 InputServlet 添加如下两个成员变量：

```

protected PaymentRemoteHome home;
protected PaymentRemote spayment;

```

(11) 再新建一个 Servlet，名称为 SearchServlet，URL 映射为 /SearchServlet。

(12) 用方法添加向导给 SearchServlet 添加两个方法，名称分别为 “lookupHome” 和 “forward”，这两个方法和 InputServlet 的完全相同，这里不再赘述。

(13) 在 SearchServlet 的 processRequest 方法中添加如下代码：

```

Vector result = new Vector( );
request.setCharacterEncoding("GBK");
String cusname = (String)request.getParameter("customerName");
cusname = new String(cusname.getBytes(), "ISO-8859-1");

```

```

home = lookupHome();
try{ spayment = home.create();
}catch(CreateException ce){
    ce.printStackTrace();
}

result = spayment.search(cusname);
request.setAttribute("result",result);
forward (request,response,"/searchresult.jsp");

```

- 此方法的功能是根据请求中用户名称的信息调用 EJB 在数据库中进行查询。
- 把查询的结果封装到 Vector 对象中转发给 searchresult.jsp 页面以显示查询结果。

(14) 在“SearchServlet”中添加如下包的引用：

```

import java.util.*;
import javax.naming.*;
import javax.rmi.*;
import javax.ejb.*;
import org.netbeans.app.StatelessBeanExample.*;

```

(15) 用字段添加向导给 SearchServlet 添加如下两个成员变量：

```

protected PaymentRemoteHome home;
protected PaymentRemote spayment;

```

完成了上述步骤，就成功地完成了 Servlet 的开发。

20.6.5 向类路径中添加并配置 EJB 模块

上一节开发的两个 Servlet 都通过 JNDI 技术调用了 EJB 模块的远程方法或 create 方法，并返回了相应的值（对象类型或远程接口类型）。因此如果要使上个小节开发的 Servlet 能够成功地编译，需要将包含 EJB 模块的 jar 文件添加到 Servlet 的类路径中。按照如下步骤完成 jar 文件的添加与配置。

(1) 用鼠标右键单击项目窗口中的“StatelessBeanExample-WebModule”/“Libraries”节点，选择右键菜单中的“Add JAR/Folder...”选项，系统会弹出添加 jar 文件或目录向导窗口，如图 20-21 所示。

(2) 在向导窗口中选择在上一小结生成的 EJB 模块 jar 文件，单击“打开”按钮完成添加。添加成功后会在“Libraries”节点出现一个“StatelessBeanExample-EJBModule.jar”子节点，如图 20-22 所示。

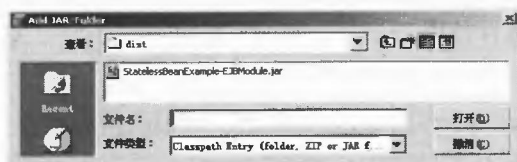


图 20-21 添加 jar 文件或目录向导窗口

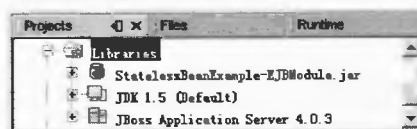


图 20-22 添加成功后的项目窗口

(3) 在项目窗口中选中 Libraries 节点，单击鼠标右键，在弹出的菜单中选择“Properties”选项，系统将弹出模块属性配置窗口，如图 20-23 所示。

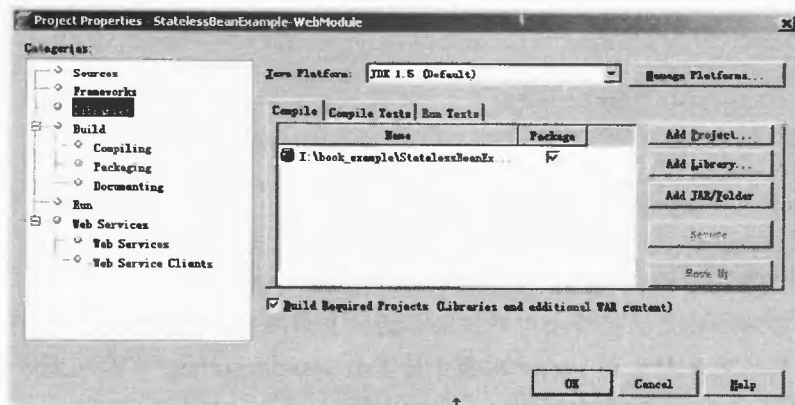


图 20-23 模块属性配置窗口

(4) 选择“Categories”列表中的“Libraries”选项，在窗体的右侧选择“Compile”标签。设置列表中“/./StatelessBeanExample-EJBModule.jar”选项后的“Package”复选框为非选中状态，单击“OK”按钮确定。

至此，所有的开发工作都完成了，在下一小节中将介绍如何编译运行整个项目。

20.7 编译运行整个项目

编译运行整个项目的步骤如下。

(1) 用鼠标右键单击项目窗口中的 StatelessBeanExample 节点，选择右键菜单中的“Clean and Build Project”选项。此时系统会自动对整个项目进行编译打包，如果没有错误，输出窗口中将显示编译打包成功的信息，如图 20-24 所示。

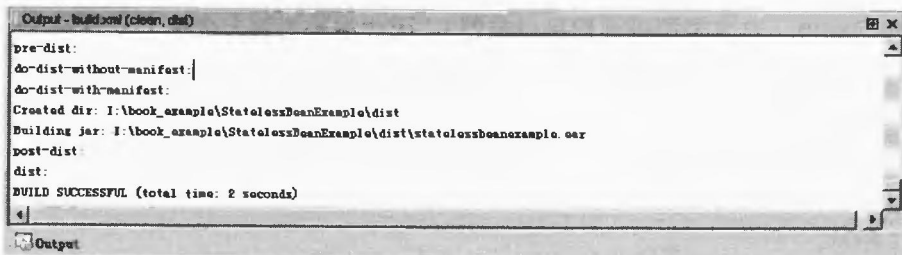


图 20-24 项目打包成功的信息

(2) 用鼠标右键单击 StatelessBeanExample 节点，在弹出的菜单中选择“Run Project”选项。经过一段时间的部署之后系统会自动弹出浏览器窗口，如图 20-1 所示。这时项目已经成功运行，读者可以按照 20.2 节中介绍的内容进行使用。

20.8 小结

在本章中介绍了无状态会话 Bean 的开发和使用，可以体会到使用 NetBeans 开发无状态会话 Bean 是非常方便的，省去了开发人员编写很多配置文件和脚本的工作，有效降低了开发成本，提高了开发效率。在后面的章节中将介绍其他几种 EJB 的开发。

第 21 章

有状态会话 Bean——实现购物车

在本章中将向读者介绍如何使用 NetBeans 进行有状态会话 Bean 的开发。首先简单介绍有状态会话 Bean，接着用一个具体的例子进行详细介绍。

21.1 有状态会话 Bean 简介

有状态会话 Bean 相当于客户的一个代理，可以控制任务流（业务过程）、填补实体 Bean 的数据，表示与业务逻辑之间的空白。有状态会话 Bean 通常还用于实体 Bean 之间的交互，另外还可以完成复杂的 Bean 处理。


每个有状态会话 Bean 在 Bean 的生命周期中都仅用于一个客户。有状态会话 Bean 不会向无状态会话 Bean 那样在 EJB 对象间交换，或保存在实例池中。一旦一个有状态会话 Bean 得到实例化，并分配给一个 EJB 对象，那么在整个的生命周期中，其将仅作用于该 EJB 对象。如果满足任意以下条件，则使用有状态会话 Bean 比较合适。

- Bean 的状态描述了 Bean 与特定客户程序的交互。
- Bean 需要保留有关客户程序的信息，且保留期限必须跨越多次方法调用。
- Bean 担负着客户程序到应用其他组件之间的中间人的角色，为客户程序提供一个简化的服务视图。
- Bean 管理着多个 EJB 的工作流程。

21.2 项目功能简介

本章中将向读者展示 NetBeans 结合 JBoss 开发一个有状态会话 Bean，并通过 Web 应用调用该有状态会话 Bean。在本例中有状态会话 Bean 担当的是购物车的角色，负责向购物车中添加用户想要购买的碟片，以及从购物车中删除用户不想购买的碟片。购买碟片界面如图 21-1 所示。

图 21-1 所示窗口中列出了可供用户购买的碟片的名称、价格以及简单的介绍。如果用户想

要购买某张碟片，只需单击相应商品的 购买按钮即可，此时该碟片会自动被添加到购物车中。

在如图 21-1 所示的窗口中选择一定商品后，单击“购物车”链接，则在新窗口中显示购物车中的碟片，如图 21-2 所示。

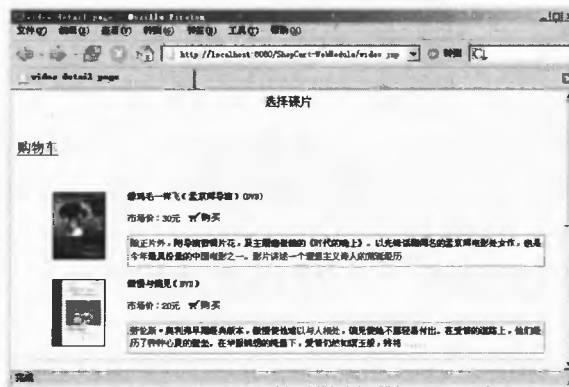


图 21-1 选择碟片页面

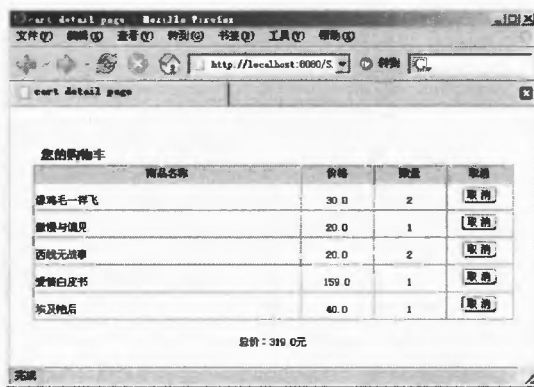


图 21-2 查看购物车窗口

图 21-2 所示窗口中列出了购物车中已有碟片的名称、价格、数量以及这些碟片的总价。单击对应碟片的“取消”按钮，可以将该碟片从购物车中删除。如单击“爱情白皮书”对应的“取消”按钮，窗口如图 21-3 所示。单击“取消”按钮以后，对应碟片将从购物车中删除。

本项目中有一个有状态会话 Bean——ShoppingCartBean，其主要用来向购物车中添加碟片，从购物车中删除碟片以及计算当前购物车中商品的总价。另外还有 1 个依赖值类表示碟片信息、2 个 JSP 页面负责表示层和 1 个 Servlet 负责控制器的功能，各个部分功能如表 21-1 所示。

表 21-1 各个组件的功能

组 件	功 能
video.jsp	负责显示碟片信息，在此页选择要购买的碟片
cart.jsp	负责显示购物车中所有碟片，以及提供对购物车的管理界面
CartServlet.java	负责处理用户的购买和取消请求
CartItem.java	依赖值类，用来表示碟片的信息

项目中各主要功能组件的关系如图 21-4 所示。

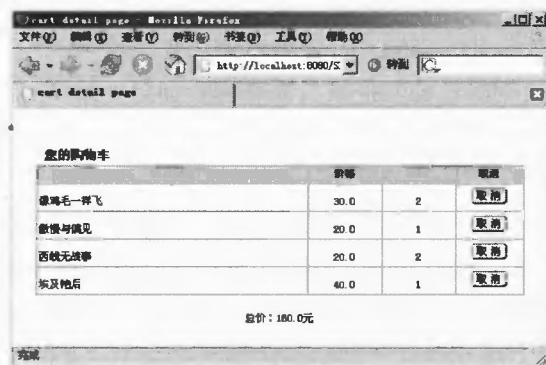


图 21-3 从购物车中删除商品

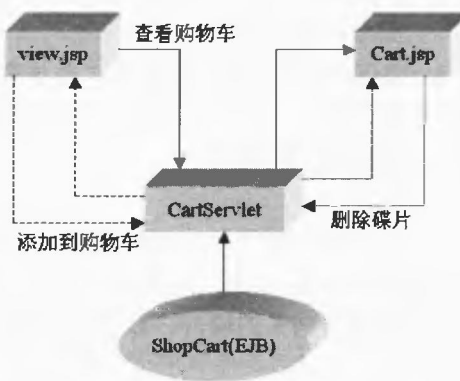


图 21-4 项目的逻辑结构图

21.3 开发 EJB 模块

从本节开始将为读者详细地介绍如何使用 NetBeans 开发有状态会话 Bean。本节主要用来开发项目的 EJB 模块部分，对于 WEB 模块的开发，将会在下一节中给出。本节主要包括如下内容：

- 有状态会话 Bean 的创建与依赖值类的开发；
- 业务代码的开发。

21.3.1 有状态会话 Bean 的创建与依赖值类的开发

按照如下步骤完成 EJB 模块的开发。

(1) 首先创建一个名称为 ShopCart 的“Enterprise Application”项目。

(2) 在项目窗口中展开 ShopCart 节点，项目窗口中内容如图 21-5 所示。

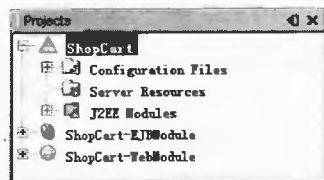


图 21-5 创建项目后的项目窗口

(3) 向 ShopCart-EJBModule 中添加包，在如图 21-5 所示的窗口中用鼠标右键单击 ShopCart-EJBModule 节点，选择右键菜单中的“New”/“Java Package”选项，出现如图 21-6 所示的窗口。

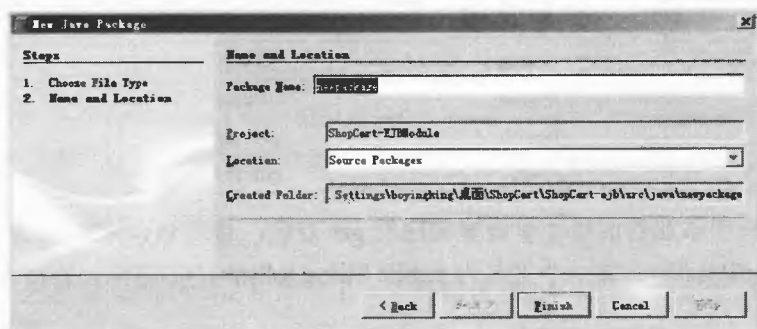


图 21-6 向 EJB 模块中添加包

(4) 可以在如图 21-6 所示的窗口中设置添加包的名称与位置。在此将添加包的名称设置为 shop.cart，单击“Finish”按钮完成添加。

(5) 在项目窗口中用鼠标右键单击“ShopCart-EJBModule”/“Source Package”节点下的 shop.cart 节点，依次选择右键菜单的“New”/“Session Bean”选项，出现如图 21-7 所示的窗口。

说明

读者对此窗口可能并不陌生，其与创建无状态会话 Bean 的窗口相同。不同的是，在此项目中创建的是有状态会话 Bean，因此要将“Session Type”的值设置为 Stateful。另外将有状态会话 Bean 的名称设置为 ShoppingCartBean，并选中“Remote”选项。单击“Finish”按钮完成创建。

(6) 在项目窗口中依次展开“ShopCart-EJBModule”/“Source Package”/“shop.cart”节点，此时项目窗口中内容如图 21-8 所示。

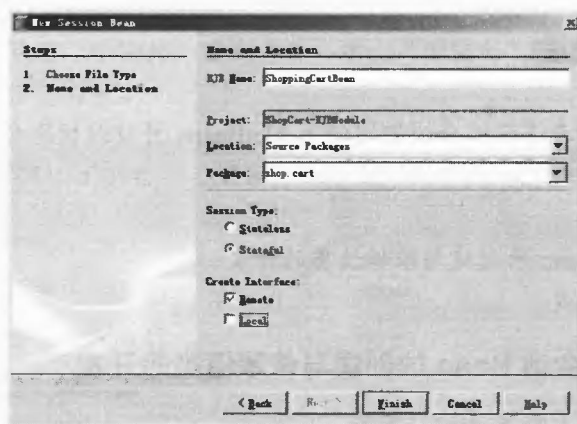


图 21-7 创建会话 Bean 窗口

(7) 从图 21-8 中可以看到, 添加有状态会话 Bean 以后, NetBeans 自动生成了相应的 Home 接口, Remote 接口以及一个业务接口。

(8) 向 shop.cart 包中添加一个名称为 CartItem 的 JavaClass, 该类是 ShopCartBean 所使用的一个依赖值类。

(9) 按如表 21-2 所示向 CartItem 中添加成员变量。

表 21-2 添加成员变量的属性及说明

变量名称	变量类型	访问限制修饰符	初始值
id	String	private	null
title	String	private	null
price	double	private	null
qty	int	private	null

(10) 为上一步添加的成员变量设置相应的 get 方法, 并为 qty 添加 set 方法。

(11) 向 CartItem 中添加一个签名为 public String toString()的方法, 并将下列代码添加到该方法中:

```
String result = id + "-->" + title + "-->" + price+"-->"+qty;
return result;
```

(12) 为 CartItem 添加一个有参数的构造器。在项目窗口中依次选择“ShopCart-EJBModule”/“Source Packages”/“shop.cart”/“CartItem.java”/“CartItem”/“Constructors”节点, 单击鼠标右键, 在弹出的菜单中依次选择“Add”/“Constructors”选项, 出现如图 21-9 所示窗口。

(13) 可以在如图 21-9 所示的窗口中设置构造器的访问限制修饰符以及参数。在本例中, 将访问限制修饰符设置为 public, 并按照如表 21-3 所示添加参数。

表 21-3 构造器的参数类型及名称

参数名称	参数类型	参数名称	参数类型
id	String	title	String
price	double	qty	Int

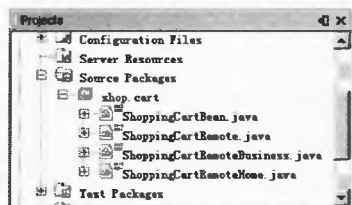


图 21-8 项目窗口中内容

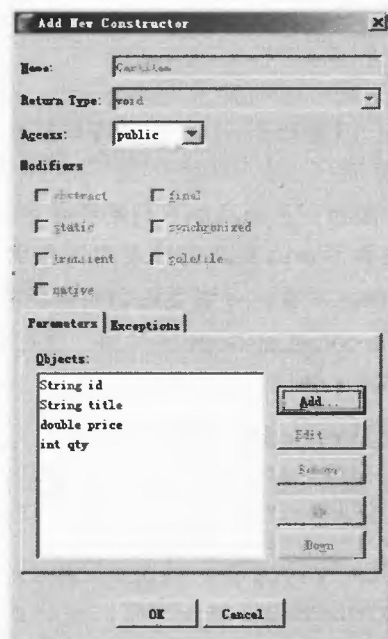


图 21-9 添加构造器窗口

(14) 将以下代码添加到步骤 (14) 所添加的构造器中:

```
this.id = id;
this.title = title;
this.price = price;
this.qty = qty;
```

完成了上述步骤, 就已经成功地完成了无状态会话 Bean 的创建与依赖值类的开发。

21.3.2 业务代码的开发

前面创建了无状态会话 Bean, 并开发了一个依赖值类。本节将介绍进行业务代码的开发。具体操作步骤如下:

(1) 向 ShoppingCartBean 类中添加一个名称为 cart、类型为 Vector、访问限制修饰符为 private 的成员变量。

(2) 将以下代码添加到 ejbCreate 方法中。

```
cart=new Vector();
```

(3) 向 ShoppingCartBean 中添加一个签名为 public void addToCart(String id, String title, double price, int quantity) 的业务方法, 并将下列代码添加到该方法中:

```
Enumeration enumr = cart.elements();
while(enumr.hasMoreElements()) {
    //检查要添加的商品是不是已经在购物车中了, 如果已经在购物车中, 则将该商品的数量加 1
    CartItem ci = (CartItem)enumr.nextElement();
    if(ci.getId().equals(id)){
        ci.setQty(ci.getQty() + quantity);
    }
}
```

```

        return;
    }
}
//创建一个新的 CartItem 类型的对象，并把它加入到购物车中
cart.add(new CartItem(id, title, price, quantity));

```

- 该方法把一件商品加入到购物车中。
- 如果要添加的商品已经在购物车中，则将该商品的数量加 1，否则创建一个新的 CartItem 对象，并将其添加到购物车中。

(4) 向 ShoppingCartBean 中添加一个签名为 `public boolean removeFromCart(String id)` 的业务方法，并将下列代码添加到该方法中：

```

for(int i=0;i<cart.size();i++) {
    CartItem ci = (CartItem)cart.elementAt( i );
    if(ci.getId().equals(id)){
        cart.remove(i);
        return true;//删除成功返回 true
    }
}
return false;// 删除失败返回 false

```


- 该方法用于从购物车中删除具有指定 ID 的商品。
- 如果购物车中存在该商品，则从购物车中删除，并返回 true，否则返回 false。

(5) 向 ShoppingCartBean 中添加一个签名为 `public double getBalance()` 的业务方法，将下列代码添加到该方法中：

```

Enumeration enumr = cart.elements();
double cartbalance = 0.0;
while(enumr.hasMoreElements()){
    CartItem ci = (CartItem)enumr.nextElement();
    cartbalance = cartbalance + ci.getQty() * ci.getPrice();
}
return cartbalance;

```

 **说明** 该方法用于计算所购买的所有商品的总价，并返回总价。

(6) 向 ShoppingCartBean 中添加一个签名为 `public Vector getCart()` 的业务方法，该方法用于返回当前的购物篮的向量对象。并将下列代码添加到该方法中：

```
return cart;
```

(7) 向 ShoppingCartBean 中添加一个签名为 `public void clearCart()` 的业务方法，该方法用于清空购物篮。并将下列代码添加到该方法中：

```
cart.clear();
```

(8) 双击项目窗口中“ShopCart-EJBModule”/“Configuration Files”/“jboss.xml”节点，在编辑器中把 jboss.xml 文件的内容替换为如下内容：


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC
"-//JBoss//DTD JBOSS 4.0//EN"
"http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>ShoppingCartBean</ejb-name>
      <jndi-name>ShoppingCartJndi</jndi-name>
    </session>
  </enterprise-beans>
</jboss>
```

- <ejb-name>标签指出 EJB 的名称，该名称要与无状态会话 Bean 类的名称相同。
- <jndi-name>指出 EJB 的 JNDI 名称。

(9) 用鼠标右键单击项目窗口中的 ShopCart-EJBModule 节点，在弹出的菜单中选择“Clean and Build Project”选项，编译并打包 EJB 模块。如果编译打包成功，输出窗口中将显示成功的信息，如图 21-10 所示。

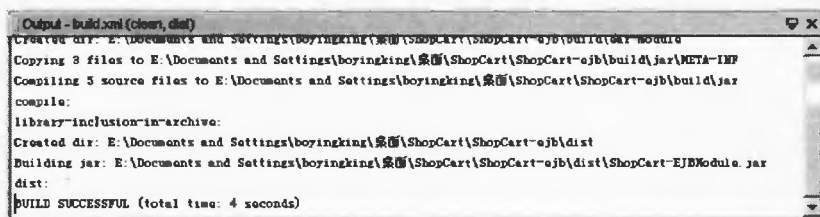


图 21-10 编译打包成功信息的输出窗口

21.4 开发 Web 模块

在上一节中已经开发了负责业务逻辑的 EJB 模块，在本小节中将介绍如何开发和终端用户交互的 Web 模块。Web 模块中的组成部分已经在表 21-1 中给出，下面将详细介绍 Web 模块的开发过程。

21.4.1 控制器 Servlet 的开发

本节将开发本例中用到的 Servlet，具体操作步骤如下。

- (1) 将 ShopCart-EJBModule.jar 文件添加到当前 Web 模块路径中，添加方法及注意事项可以参照 20.6.5 小节。
- (2) 向 ShopCart-WebModule 中添加一个名称为“shop.cart”的包。
- (3) 向 shop.cart 包中添加一个名称为 CartServlet，URL 映射值为 /CartServlet 的 Servlet。
- (4) 在编辑器窗口中打开 CartServlet，向其中添加一个签名为 private ShoppingCartRemoteHome lookupHome() 的方法，并将下列代码添加到该方法中：

```
ShoppingCartRemoteHome myHome = null;
try {
```

```

        Hashtable ht = new Hashtable();
        ht.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
        ht.put(Context.PROVIDER_URL, "localhost:1099");
        Context ctx = new InitialContext(ht);
        Object home = ctx.lookup("ShoppingCartJndi");
        //把远程对象引用转换成 Home 句柄
        myHome = (ShoppingCartRemoteHome)
            PortableRemoteObject.narrow(home, ShoppingCartRemoteHome.class);
    } catch (Exception e) {
        System.out.println("The client was unable to lookup the EJBHome.");
        e.printStackTrace();
    }
    return myHome;

```

- 该方法的主要功能是获取并返回“ShoppingCartBean”EJB的Home句柄。
- ctx.lookup("ShoppingCartJndi")通过前面在jboss.xml文件中配置的EJB JNDI名称来获取远程对象引用。

(5) 向 CartServlet 中添加一个签名为 protected void forward(HttpServletRequest request, HttpServletResponse response, String url) throws ServletException, IOException 的方法, 并将下列代码添加到该方法中:

```

RequestDispatcher dispatcher = request.getRequestDispatcher(url);
dispatcher.forward(request, response);

```

说明 此方法的功能是把请求进行转发。

(6) 按如表 21-4 所示向 CartServlet 中添加成员变量。

表 21-4 添加成员变量

变量名称	变量类型	访问限制修饰符	初始值
home	ShoppingCartRemoteHome	private	null
shoppingCart	ShoppingCartRemote	private	null

(7) 向 CartServlet 中添加一个签名为 public void init() throws ServletException 的方法, 并将下列代码添加到该方法中:

```

home = lookupHome();
super.init();

```

(8) 向 processRequest 方法的方法体内添加如下代码:

```

//能处理中文参数乱码问题
request.setCharacterEncoding("ISO-8859-1");
//得到参数 a 的数值
String action = (String)request.getParameter("a");
HttpSession session = request.getSession(true);
//如果用户不是第一次访问, 通过 Session 让用户使用已有的购物车实例, 否则建一个新的

```

```

if(session.getAttribute("cart") == null){
    try{
        shoppingCart = home.create();
        session.setAttribute("cart",shoppingCart);
    }catch(Exception ce){
        ce.printStackTrace();
    }
}
else{
    shoppingCart = (ShoppingCartRemote)session.getAttribute("cart");
}
//处理各种请求
if(action.equals("get")){//处理请求购物车的请求
    //跳转页面
    forward(request,response,"/cart.jsp");
}else if (action.equals("remove")){//处理删除购物车中商品的请求
    String id = (String) request.getParameter("id");
    shoppingCart.removeFromCart(id);
    forward(request,response,"/cart.jsp");
}else if(action.equals("add")){//处理往购物车中加商品的请求
    String id = (String) request.getParameter("id");
    String title = (String) request.getParameter("title");
    title=new String(title.getBytes("ISO-8859-1"),"GBK");
    double price = Double.parseDouble(request.getParameter("price"));
    shoppingCart.addToCart(id,title,price,1);
    forward(request,response,"/video.jsp");
}

```

完成了上述步骤，就已经完成了 Servlet 的开发。

21.4.2 开发商品的显示与购买页面

本节将开发此例中用到的商品的显示与购买页面——video.jsp, 按照如下步骤完成该 JSP 的开发。

- (1) 向 ShopCart-WebModule 中添加一个名称为 video 的 JSP。
- (2) 在编辑器窗口中打开该文件，删除其中自动生成的代码，并添加如下代码：

```

<%@ page contentType="text/html;charset=GBK"%>
<html>
<head>
<title>video detail page</title>
<link href="<%=request.getContextPath()%>/images/dvd.css" rel="stylesheet" type="text/css">
</head>
<body></FONT></TD>
</TR>
<p align="center" class="bordercd">

```

```

        <span class="hangju"><font color="#000000">选择碟片</font></span></p>
        <p>&nbsp; </p>
        <a href="<%=request.getContextPath()%>/CartServlet?a=get" target=
        "_blank">购物车</a>
        <p>&nbsp; </p>
        <TABLE align=center border=0 cellpadding=0 cellspacing=0 width="95%">
        <TBODY>
        <TR>
        <TD vAlign=top width="14%"> <DIV align=center><IMG
        border=1 height=90
        src="<%=request.getContextPath()%>/images/tn_jimaodvd.jpg"
        width=70> </DIV></TD>
        <TDwidth="60%"><FONTcolor=#000000>像鸡毛一样飞 (孟京辉导演) (DVD) <BR>
        <BR>
        市场价: 30 元 <a href=
        "<%=request.getContextPath()%>/CartServlet?a=add&id=101&price=30&titl
        e=像鸡毛一样飞">
        <IMG border=0 height=12 src="<%=request.getContextPath()%>/images/buy.
        gif" width=43></a> <BR>
        <BR>
        <TABLE border=0 class=newjoyo_vcd1 width="100%">
        <TBODY>
        <TR>
        <TD><FONT
        color=#000000>除正片外, 附导演剪辑片花, 及主题曲崔健的《时代的
        上》。以先锋话剧闻名的孟京辉电影处女作, 也是今年最具份量的中国电影之一。影片讲述一个理想
        主义诗人的荒诞经历……</FONT></TD>
        </TR>
        </TBODY>
        </TABLE>
        </FONT></TD>
        </TR>
        </TBODY>
        </TABLE>
        <HR color=#cccccc noShade SIZE=1 width="95%">
        <TABLE align=center border=0 cellpadding=0 cellspacing=0 width="95%">
        <TBODY>
        <TR>
        <TD vAlign=top width="14%"> <DIV align=center><IMG
        border=1 height=90 src="<%=request.getContextPath()%>/images/
        tn_aomanyudvd.jpg" width=70> </DIV></TD>
        <TD width="60%"><FONT color=#000000>傲慢与偏见 (DVD) <BR>
        <BR>
        市场价: 20 元 <ahref="<%=request.getContextPath()%>/CartServlet?a
        =add&id=102&price=20&title=傲慢与偏见"><IMG
        border=0 height=12 src="<%=request.getContextPath()%>/images/
        buy.gif" width=43></a> <BR>

```

```

<BR>
<TABLE border=0 class=newjoyo_vcd1 width="100%">
  <TBODY>
    <TR>
      <TD><FONT
        color=#000000>劳伦斯·奥利弗早期经典版本，傲慢使他难以与人相处，
        偏见使她不愿轻易付出。在爱情的道路上，他们经历了种种心灵的壁垒。在华服锦绣的掩盖下，爱情
        仍然如璞玉般，终将……</FONT></TD>
      </TR>
    </TBODY>
  </TABLE>
  </FONT></TD>
</TR>
</TBODY>
</TABLE>
<HR color=#cccccc noShade SIZE=1 width="95%">
<TABLE align=center border=0 cellPadding=0 cellSpacing=0 width="95%">
  <TBODY>
    <TR>
      <TD height=2 width="14%"><FONT size=-1>&nbsp;</FONT></TD>
      <TD height=2 width="60%">&nbsp;</TD>
    </TR>
    <TR>
      <TD vAlign=top width="14%"> <DIV align=center><IMG
        border=1 height=90 src="<%=request.getContextPath()%>/images
        /tn_xixianwuvcd.jpg" width=70> </DIV></TD>
      <TD width="60%"><FONT color=#000000>西线无战事 (DVD) <BR>
      <BR>
      市场价: 20 元 <a href="<%=request.getContextPath()%>/CartServlet
      ?a=add&id=103&price=20&title=西线无战事"><IMG
        border=0 height=12 src="<%=request.getContextPath()%>/images/
        buy.gif" width=43></a> <BR>
      <BR>
      <TABLE border=0 class=newjoyo_vcd1 width="100%">
        <TBODY>
          <TR>
            <TD><FONT
              color=#000000>被称为“影史上最伟大的反战电影”，获奥斯卡最佳影片、
              最佳导演奖。本片以卓越的摄影技巧，写实的风格，把战火阵地、冲锋、肉搏，以及残酷恐怖的野蛮
              凶杀等镜头，拍得……</FONT></TD>
            </TR>
          </TBODY>
        </TABLE>
        </FONT></TD>
      </TR>
    </TBODY>
  </TABLE>

```



```

<HR color=#cccccc noShade SIZE=1 width="95%">
<TABLE align=center border=0 cellPadding=0 cellSpacing=0 width="95%">
  <TBODY>
    <TR>
      <TD height=2 width="14%"><FONT size=-1>&nbsp;</FONT></TD>
      <TD height=2 width="60%">&nbsp;</TD>
    </TR>
    <TR>
      <TD vAlign=top width="14%"> <DIV align=center><IMG
        border=1 height=90 src="<%=request.getContextPath()%>/images/
tn_aijiyangdvd.jpg" width=70> </DIV></TD>
      <TD width="60%"><FONT color=#000000>埃及艳后 (双 DVD) <BR>
        市场价: 40 元 <a href="<%=request.getContextPath()%>/CartServlet?a=
add&id=104&price=40&title=埃及艳后"><IMG
        border=0 height=12 src="<%=request.getContextPath()%>/images/
buy.gif" width=43></a> <BR>
        <BR>
        <TABLE border=0 class=newjoyo_vcd1 width="100%">
          <TBODY>
            <TR>
              <TD><FONT
                color=#000000>如果去除货币贬值和物价因素,这部电影是历史上最昂贵
的电影,远胜《泰坦尼克》,它几乎使福克斯公司破了产。在当时电脑特技不发达的情况下,所有的
宏大场面和镜头全部是……</FONT></TD>
            </TR>
          </TBODY>
        </TABLE>
      </FONT></TD>
    </TR>
  </TBODY>
</TABLE>
<HR color=#cccccc noShade SIZE=1 width="95%">
<TABLE align=center border=0 cellPadding=0 cellSpacing=0 width="95%">
  <TBODY>
    <TR>
      <TD height=2 width="14%"><FONT size=-1>&nbsp;</FONT></TD>
      <TD height=2 width="60%">&nbsp;</TD>
    </TR>
    <TR>
      <TD vAlign=top width="14%"> <DIV align=center><IMG
        border=1 height=90 src="<%=request.getContextPath()%>/images/
tn_aiqingbpsdvd.jpg" width=70> </DIV></TD>
      <TD width="60%"><FONT color=#000000>爱情白皮书 (DVD) <BR>
        <BR>
        市场价: 159 元 <a href="<%=request.getContextPath()%>/CartServlet?

```



```

a=add&id=105&price=159&title=爱情白皮书"><IMG border=0 height=12
src="<%=request.getContextPath()%>/images/buy.gif" width=43></a><BR>
<BR>
<TABLE border=0 class=newjoyo_vcd1 width="100%">
<TBODY>
<TR>
<TD><FONT
color=#000000>柴门文首次同意授权海外拍摄的作品，凤凰格力剧场热
播，《流星花园》“小优”杨丞琳担纲主演！此剧改编自《东京爱情故事》作者、日本漫画大师柴门
文另一畅销名作，1……</FONT></TD>
</TR>
</TBODY>
</TABLE>
</FONT></TD>
</TR>
</TBODY>
</TABLE>
</body>
</html>

```

- 该 JSP 用来显示已有的碟片信息，在此页面中选择购买的商品。
- `<%=request.getContextPath()%>` 用来获取当前的上下文路径。

(3) 将 video.jsp 设置为 ShopCart-WebModule 的欢迎页面。完成了上述步骤，就已经成功地完成了 video.jsp 的开发。

21.4.3 开发购物车

本小节将开发本项目中用到的购物车页面——cart.jsp，按照如下步骤完成该 JSP 的开发。

- (1) 向 ShopCart-WebModule 中添加一个名称为 cart 的 JSP。
- (2) 在编辑器窗口中打开该文件，删除其中自动生成的代码，并添加如下代码：

```

<%@ page import="shop.cart.*,java.util.*" %>
<%@ page contentType="text/html;charset=GBK"%>
<html>
<head>
<title>cart detail page</title>
<link href="<%=request.getContextPath()%>/images/dvd.css" rel=
"stylesheet" type="text/css">
</head>
<body>
<p>&nbsp;</p><table align=center border=0 cellpadding=3 cellspacing=0
width=561>
<tbody>
<tr>
<td width="500" valign=top><table width=100% height="100%" border=
0 align=center cellpadding=2 cellspacing=0>
<tbody>

```

精通 NetBeans——Java 桌面、Web 与企业级程序开发详解 472

```

<td height=25 width=68>
    <font color=#000000><%=ci.getQty
() %> </font></td>

    <td height=25 width=62><font color=
#000000><a href="<%=request.getContextPath() %>/CartServlet?a=remove&id=<%=
ci.getId() %>"></a> </font></td>
    </tr>
    <%}%>
</tbody>
</table></td>
</tr>
<tr align=middle>
    <td height=40><br> <table border=0 cellpadding=0
cellspacing=0 width="85%">
        <tbody>
            <tr>
                <td align=middle width="25%">总价:<%=cart.
getBalance() %>元</td>
            </tr>
        </tbody>
    </table>
</td>
</tr>
</tbody>
</table>
</body>
</html>

```

该 JSP 用来负责显示购物车中所有碟片，以及提供对购物车的管理界面。完成了上述步骤，就已经成功地完成了 cart.jsp 的开发。

21.5 编译运行整个项目

编译运行整个项目的步骤如下。

(1) 用鼠标右键单击项目窗口中的 ShopCart 节点，选择右键菜单中的“Clean and Build Project”选项。此时系统会自动对整个项目进行编译打包，如果没有错误，输出窗口中将显示编译打包成功的信息，如图 21-11 所示。

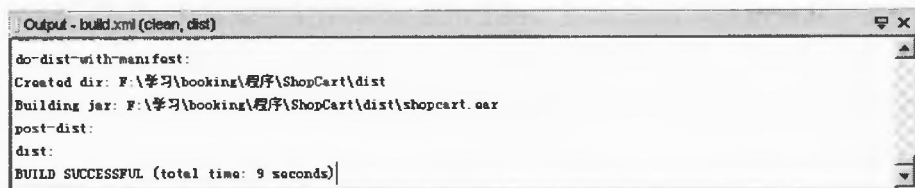


图 21-11 项目打包成功的信息

(2) 用鼠标右键单击项目窗口中的 ShopCart 节点, 选择右键菜单中的 “Run Project” 选项。经过一段时间的部署之后系统会自动弹出浏览器窗口, 如图 21-1 所示。这时项目已经成功运行, 读者可以按照 21.2 节中介绍的内容进行使用。

21.6 小结

在本章中介绍了有状态会话 Bean 的开发和使用, 可以体会到使用 NetBeans 开发有状态会话 Bean 是非常方便的, 省去了开发人员编写很多配置文件和脚本的工作, 从而有效地降低了开发成本, 提高了开发效率。在后面的章节中将继续介绍其他几种 EJB 的开发。

第 22 章

开发 CMP 实体 Bean——图书信息管理

本章将向读者介绍如何使用 NetBeans 开发 CMP。首先简单介绍了 CMP，接着通过一个例子详细地讲述了如何使用 NetBeans 开发 CMP。主要包括如下内容：

- CMP 简介；
- 项目功能介绍；
- 项目的结构及模块功能说明；
- 数据库准备工作；
- 开发 CMP 模块；
- 开发 Web 模块；
- 编译运行整个项目。

22.1 CMP 简介

本节将对与 CMP 有关的内容进行简单的介绍，主要包括如下内容：

- 实体 Bean 的简单介绍；
- CMP 的简单介绍；
- 持久性类的简单介绍。

22.1.1 实体 Bean 的简单介绍

想要理解 CMP，首先要对实体 Bean 有所了解。实体 Bean 表示数据库中的数据，对实体 Bean 的修改将使数据库内容随之调整。实体 Bean 的使用，为开发人员提供了一种更简单的机制来访问数据库。根据管理实体 Bean 持久存储方式的不同，存在以下两种类型的实体 Bean：

- 容器管理的持久存储实体 Bean (container-managed persistence bean, CMP)；
- Bean 管理的持久存储实体 Bean (bean-managed persistence bean, BMP)。

22.1.2 CMP 的简单介绍

对于容器管理的持久存储实体 Bean（通常简称为 CMP），容器将负责实体 Bean 在事务中的登记，并负责把其状态保存在数据库中。

开发人员使用虚持久性字段和关系字段来描述一个实体 Bean 的属性和关系。之所以称为虚持久性字段，主要因为以下两点原因。

- Bean 的开发人员并不显式地声明这些字段。
- 要在实体 Bean 类中声明抽象存取方法（get 与 set 方法）。这些方法的实现将在部署时由 EJB 开发商的容器工具生成。



说明

前面提到的关系字段和持久性字段，均是指抽象存取方法，而不是类中声明的实际字段。读者可以自行查阅一些资料，或从后面的实例中学习。

22.1.3 持久性类的简单介绍

在 CMP 中，容器的主要职责是生成抽象实体 Bean 的具体实现。由容器工具生成的具体类称为持久性类。持久性类实例的主要作用是在运行时，利用容器在实体 Bean 和数据库之间读写数据（即以实体 Bean 和数据库作为数据读写的源或目标）。



说明

一旦生成了持久性类，就可以将其部署到 EJB 容器中。容器将通知持久性实例（持久性类的实例），何时最适于读写数据库数据。

由于 EJB 容器会在部署 CMP 时自动生成持久性类（包括数据库访问逻辑），因此 Bean 开发人员不必自行编写数据库访问代码。作为 EJB 开发人员，在使用 CMP 时，不必自行编写数据库访问代码。

对于 Bean 管理的持久存储 Bean (BMP)，开发人员必须显式地将持久存储逻辑写入 Bean 类中。因为 BMP 的应用越来越少，本书不会对 BMP 进行详细的介绍，有兴趣的读者可以查阅相关的资料。

22.2 项目功能介绍

在本节中将介绍如何使用 NetBeans 结合 JBoss 开发一个 CMP，并通过 Web 应用调用该 CMP。在本例中，CMP 负责的功能主要是向数据库中添加书籍、删除书籍、修改指定编号的书籍的信息以及查看所有书籍的信息。项目运行时添加书籍的界面如图 20-1 所示。

可以在如图 22-1 所示的窗口中填写要添加的图书的信息，包括图书编号、图书名称以及图书价格，设置完成后，单击“添加”按钮提交数据。此时数据会被写到数据库中，浏览器中内容如图 22-2 所示。

从图中可以看到，添加图书成功以后，出现“添加成功”的提示信息。查看数据库，可以发现相应的记录确实被添加到数据库中。按如表 22-1 所示再向数据库中添加 4 本图书。

表 22-1

添加图书的信息

图书编号	图书名称	图书价格
1000002	Java 算法	49.5
1000003	J2EE 核心技术	58.4
1000004	Oracle 数据库编程宝典	88.2
1000005	J2EE1.4 标准教材	100.0

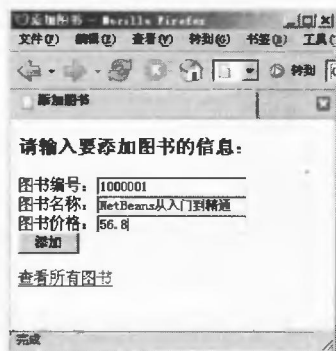


图 22-1 添加图书窗口

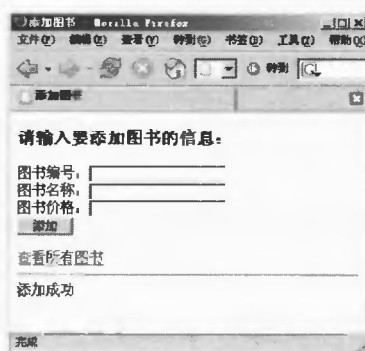


图 22-2 添加成功窗口

添加完成以后，单击“查看所有图书”链接，此时浏览器中内容如图 22-3 所示。

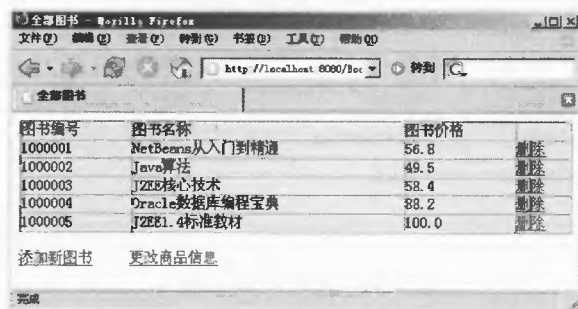


图 22-3 查看图书信息

图 22-3 所示窗口列出了数据库中已有图书的信息，包括图书编号、图书名称、图书价格。单击对应商品的“删除”连接，可以将该商品从数据库中删除。如单击编号为 1000005 的商品所对应的“删除”链接后，窗口内容如图 22-4 所示。

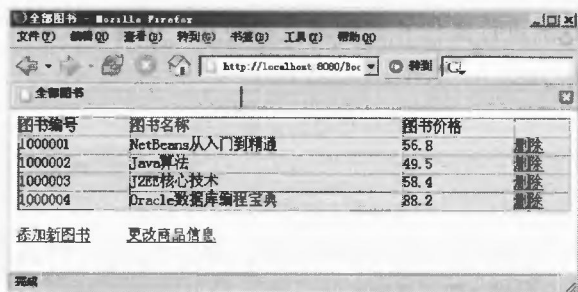


图 22-4 删除商品后的窗口

从图 22-4 中可以看到，对应的商品确实被删除了。单击“添加新图书”链接，转到图

22-1 所示窗口。单击“更改商品信息”链接，浏览器中内容如图 22-5 所示。

可以在如图 22-5 所示的窗口中修改指定编号的图书的信息。如修改编号为 1000002 的图书信息，新图书名称为“Java 网络编程”，新图书价格为 85.3。单击“更改”按钮后，浏览器中内容如图 22-6 所示。

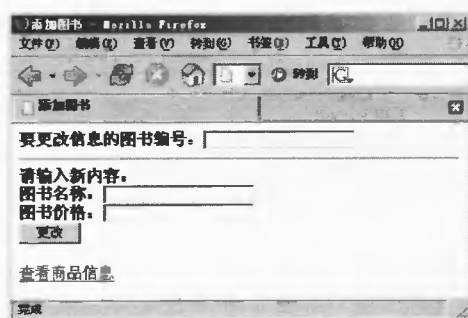


图 22-5 更改信息窗口

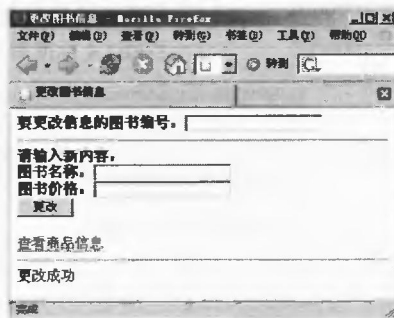


图 22-6 修改成功窗口

从图 22-6 中可以看到，修改成功后，出现“更改成功”的提示信息。此时单击“查看商品信息”链接，窗口如图 22-7 所示。

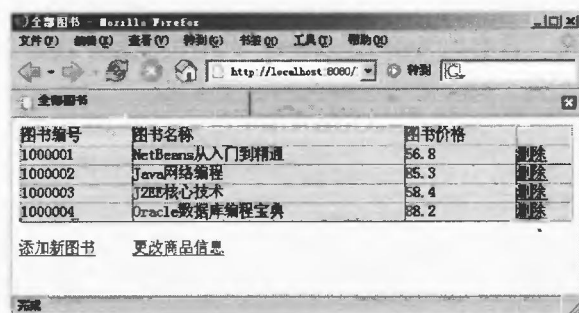


图 22-7 修改信息后窗口

从图中可以看到，编号为 1000002 的商品信息确实被修改了。

22.3 项目的结构及模块功能说明

本项目中包含一个 CMP，其负责与数据库的交互。包含一个 Servlet，用来根据客户端的请求调用 CMP 的相应方法进行操作并控制请求的转发。包含 3 个 JSP 文件，负责数据的显示。各自功能的详细说明如表 22-2 所示。

表 22-2 模块功能的详细说明

模块名称	功能说明
BookShopBean	CMP，用来进行与数据库有关的操作
BookShopServlet	根据从客户端接收的请求，调用相应的方法响应用户请求
addbook.jsp	添加图书界面
changebook.jsp	修改图书信息界面
showadddelete.jsp	负责显示图书信息，也可以在该页删除相应的图书

项目中所有 JSP、Servlet 以及 CMP 的关系如图 22-8 所示。

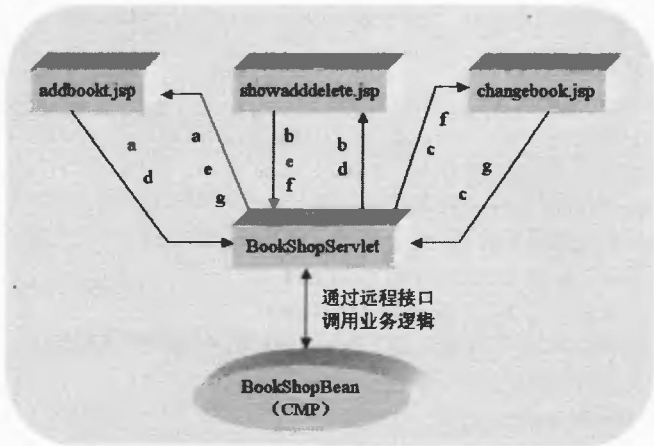


图 22-8 模块的逻辑结构图

表 22-3 给出了各个编号所对应的动作类型。

表 22-3 动作编号对照表

动作编号	动作内容	动作编号	动作内容
a	添加图书	b	删除图书
c	修改图书信息	d	查看图书信息
e	转到添加图书页面	f	转到更改图书信息页面
g	转到添加图书页面		

22.4 数据库准备工作

在开发本项目时需要用到数据库，在开发项目之前，首先要进行一些数据库的准备工作。在本项目中选用开源数据库 MySQL 作为项目的后台数据库，在数据库中创建一个名称为 bookshop 的表，表中各个字段的名称和类型如表 22-4 所示。

表 22-4 bookshop 的各个字段

字段名称	字段类型	约束条件
id	VARCHAR(20)	PRIMARY KEY
name	VARCHAR(30)	NOT NULL
price	NUMBER(6,2)	NOT NULL

关于创建表的具体步骤，读者可以参照 19.4.4 的内容完成，这里不再赘述。

在成功地创建了数据库表后，就要为 CMP 在 JBoss 中配置相应的数据源。由于 JBoss 是一个开源的应用服务器，所以其不像 WebLogic 这样的商业应用服务器有非常友好的用户配置界面，在 JBoss 中配置数据源的方式就是编写相应的 XML 配置文件。在本项目中编写一个名称为 mysql-cmp-ds.xml 的 XML 文件，并把该文件存放到 JBoss 安装目录的 server 目录下 default 目录中的 deploy 子目录中。

mysql-cmp-ds.xml 配置文件的类容如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>cmpbook</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/test</connection-url>
    <driver-class>org.gjt.mm.mysql.Driver</driver-class>
    <user-name>root</user-name>
    <password></password>
    <min-pool-size>5</min-pool-size>
    <max-pool-size>20</max-pool-size>
    <idle-timeout-minutes>0</idle-timeout-minutes>
    <track-statements/>
  </local-tx-datasource>
</datasources>
```

- <jndi-name>标签中包含的是数据源的 JNDI 名称。
- <connection-url>标签中包含的是连接 MySQL 数据库的连接字符串。
- <driver-class>标签指出连接所使用的驱动程序。
- <user-name>标签指出要连接的数据库的用户名。
- <password>标签指出要连接的数据库的密码, 如果没有密码则为空。
- <min-pool-size>指出连接池的最少连接数。
- <max-pool-size>指出连接池的最大连接数。

在编写完配置文件后, 在 NetBeans 中启动 JBoss 服务器, 测试数据源是否创建成功。如果数据源配置正确, 在启动信息窗口中就会看到数据源创建成功的信息, 如图 22-9 所示。

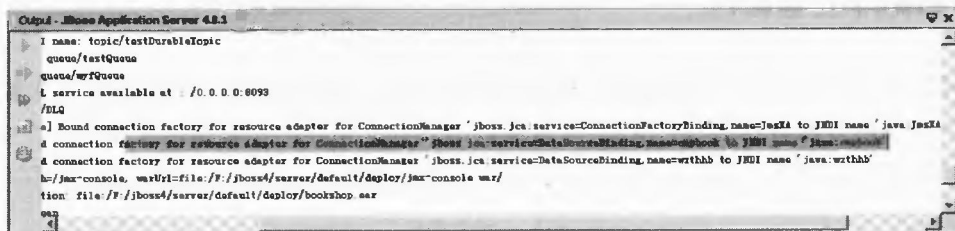


图 22-9 配置数据源成功窗口

完成了上述步骤, 就已经成功地完成了与数据库有关的准备工作。

22.5 开发 CMP 模块

在本节中将进行 CMP 模块的开发, 主要包括如下内容:

- CMP 的创建与关系字段的添加;
- 添加 create 与 find 方法;
- 配置主键与关系字段;
- 配置 CMP 与数据库映射。

22.5.1 CMP 的创建与关系字段的添加

本节将创建 CMP，并向其中添加了几个关系字段。按照如下步骤完成 CMP 的开发与关系字段的添加。

- (1) 创建一个名称为 BookShop 的 Enterprise Application。
- (2) 向 BookShop-EJBModule 中添加一个名称为 org.netbeans.bookshop 的包。
- (3) 添加实体 Bean。在项目窗口中选中“BookShop-EJBModule”/“Source Packages”/“org.netbeans.bookshop”节点，单击鼠标右键，在弹出的菜单中依次选择“New”/“Entity Bean”选项，出现如图 22-10 所示窗口。

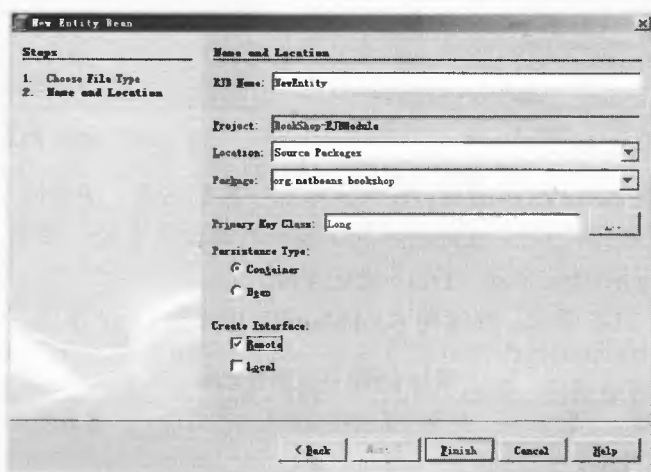


图 22-10 新建实体 Bean 窗口

- (4) 按照如表 22-5 所示对图 22-10 中的相应选项进行设置。

表 22-5

各主要选项的说明及设置值

选项名称	说明	设置值
EJB Name	要创建的实体 Bean 的名称	BookShopBean
Primary Key Class	主键类型	java.lang.String
Persistence Type	持久性类型	Container
Create Interface	创建接口	仅选中 Remote 选项（仅创建远程接口）

- (5) 设置完成以后，单击“Finish”按钮完成实体 Bean 的创建。

(6) 在项目窗口中展开“BookShop-EJBModule”/“Enterprise Beans”/“BookShopEB”节点，此时项目窗口中内容如图 22-11 所示。

(7) 从如图 22-11 所示的窗口中可以看到，NetBeans 自动添加了一个名称为 key 的关系字段。并且创建了 3 个远程方法，即 findByPrimaryKey、create 与 getKey。同时自动生成了相应的 Home 接口，Remote 接口以及一个业务接口。

(8) 向 BookShopEB 中添加 CMP 字段。在如图 22-11 所示的窗口中，用鼠标右键单击 BookShopEB 节点，选择右键菜单中的“Add”/“Add CMP Field”，出现图 22-12 所示窗口。

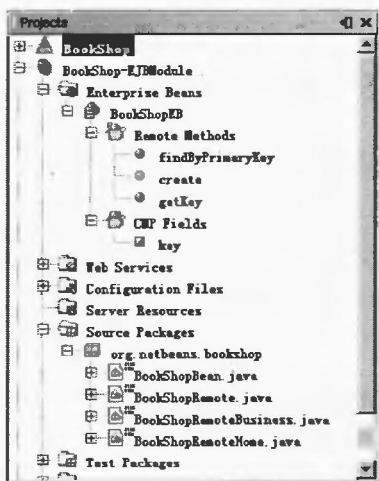


图 22-11 项目窗口内容



图 22-12 添加 CMP 字段


(9) 可以在如图 22-12 所示的窗口中设置添加字段的名称, 类型以及是否向远程接口中添加该字段的 get 或者 set 方法。在此将添加字段的名称设置为 ID, 类型为 String, 不向远程接口中添加 Getter 与 Setter, 单击“OK”按钮完成。

(10) 根据如表 22-6 所示, 继续向 BookShopEB 中添加 CMP 字段。

表 22-6

添加 CMP 字段的各选项

字段名称	类 型	是否向远程接口添加 get 方法	是否向远程接口添加 set 方法
name	String	是	是
price	Double	是	是

 **说明** 添加的 CMP 字段时, 并没有实际声明相应的字段。如果用户查看源代码会发现其中只生成了相应的 get 与 set 方法, 则这里所添加 CMP 字段即为前面所提到的虚持久性字段。

(11) 此时在编辑器窗口中打开 BookShopBean.java 文件, 会发现添加了关系字段后, 自动生成了如下代码:

```
public abstract java.lang.String getId();
public abstract void setId(java.lang.String id);
public abstract java.lang.String getName();
public abstract void setName(java.lang.String name);
public abstract java.lang.Double getPrice();
public abstract void setPrice(java.lang.Double price);
```

(12) 在编辑器中打开 BookShopRemoteBusiness.java 文件, 会发现自动生成了如下代码:

```
void setName(java.lang.String name) throws java.rmi.RemoteException;
java.lang.String getName() throws java.rmi.RemoteException;
void setPrice(java.lang.Double price) throws java.rmi.RemoteException;
java.lang.Double getPrice() throws java.rmi.RemoteException;
```

完成了上述步骤, 即完成了 CMP 的创建与关系字段的添加。

22.5.2 添加 create 与 find 方法

上面小节创建了 CMP，并向其中添加了 3 个关系字段。在本节中，将主要介绍如何向 CMP 中添加 create 与 find 方法。按照如下步骤完成 create 与 find 方法的添加。

(1) 在创建 CMP 时，容器自动生成了一个 create 方法。在此例中不会用到该方法，因此应该将该方法删除。在如图 22-11 所示的窗口中的 create 节点上单击鼠标右键，选择右键菜单中的“Delete”选项删除该 create 方法。

(2) 向 BookShopEB 中添加 create 方法。在如图 22-11 所示的项目窗口中 BookShopEB 节点上单击鼠标右键，选择右键菜单中的“Add”/“Add Create Method”选项，出现如图 22-13 所示的窗口。

(3) 将添加方法的名称设置为 create，并为该方法添加名称为 ID、name 与 price，类型分别为 String、String 与 Double 的参数，单击 OK 按钮完成添加。

(4) 用鼠标右键单击 create 节点，选择右键菜单中的“Open”选项，此时光标定位到 BookShopBean 中 ejbCreate 方法的方法体，将下列代码添加到其中：

```
setId(id);  
setName(name);  
setPrice(price);
```

(5) 向 BookShopEB 中添加 finder 方法。在项目窗口中的 BookShopEB 节点上单击鼠标右键，选择右键菜单中的“Add”/“Add Finder Method”选项，出现如图 22-14 所示的窗口。

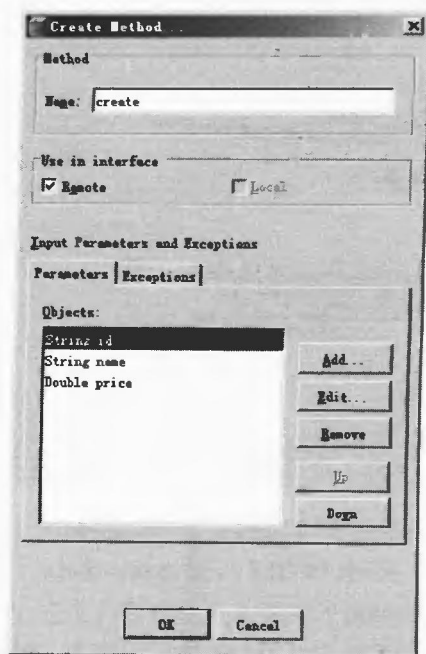


图 22-13 添加 create 方法

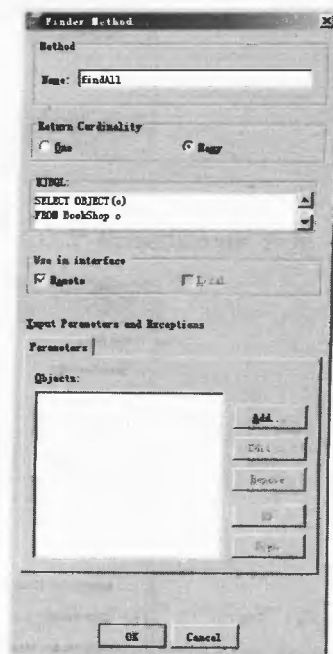


图 22-14 添加 finder 方法

(6) 可以在如图 22-14 所示的窗口中设置添加方法的名称，返回值为多个还是 1 个，并且可以自行编写 EJBQL。在此将方法的名称设置为 findAll，其余选项使用默认值。单击“OK”

按钮完成添加。添加的该方法将从数据库中查找所有的记录。

EJBQL 是从 EJB 2.0 开始引入的,是一种描述性的查询语言,类似于关系数据库中所使用的结构化查询语言。不同的是 EJBQL 按照实体 Bean 的抽象存储模式来定义,而不是根据其底层的数据库,因此具有更好的可移植性。

完成了上述步骤,就已经成功地完成了 create 和 find 方法的添加。

22.5.3 配置主键与关系字段

通过前面两个小节的工作,已经完成了该 CMP 模块代码框架的开发。在本小节中,将对该 CMP 进行配置,按照如下步骤完成配置工作。

(1) 在项目窗口中选中“BookShop-EJBModule”/“Configuration Files”/“ejb-jar.xml”节点,单击鼠标右键,在弹出的菜单中选择“Open”选项以在编辑器窗口中打开该文件。此时编辑器窗口中内容如图 22-15 所示。

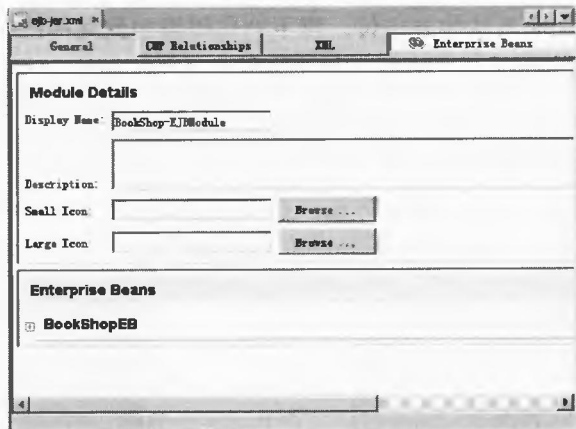


图 22-15 普通配置窗口

(2) 展开 BookShopEB 节点,如图 22-16 所示。

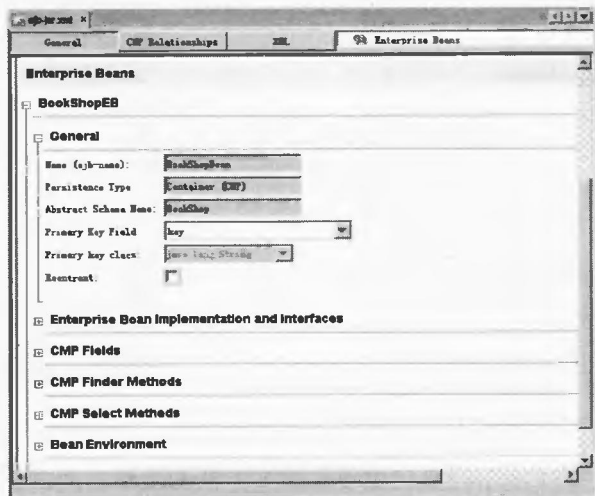


图 22-16 设置主键

(3) 将字段 ID 设置为主键。单击“Primary Key Field”文本框右侧的▼按钮，会列出当前 CMP 中已有的字段，在此选择“ID”选项，即将 ID 设置为该 CMP 的主键。

(4) 在如图 22-16 所示的窗口中展开“CMP Fields”选项，内容如图 22-17 所示。

CMP Fields						
Field Name	Type	Local C...	Local S...	Remote ...	Remote ...	Description
id	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
key	java.lang.String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
name	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
price	java.lang.Double	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

图 22-17 设置 CMP 字段

(5) 选中 key 所在行，单击“Remove”删除该字段，如图 22-17 所示。

提示 key 表示创建 CMP 时容器自动生成的字段，在此例中并没有用到。

22.5.4 配置 CMP 与数据库映射

本节将主要用来配置 CMP 的 JNDI 名称以及 CMP 与数据库的映射，按照如下步骤完成配置。

(1) 在项目窗口中打开 jboss.xml 文件，删除其中自动生成的代码，并将下列代码添加到其中：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC
    "-//JBoss//DTD JBOSS 4.0//EN"
    "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss>
    <enterprise-beans>
        <entity>
            <ejb-name>BookShopBean</ejb-name>
            <jndi-name>BookShopBeanJndiName</jndi-name>
            <method-attributes>
            </method-attributes>
        </entity>
    </enterprise-beans>
</jboss>
```

- <ejb-name>标签指出 EJB 的名称。
- <jndi-name>指出 EJB 的 JNDI 名称。

(2) 在文件窗口中依次选中“BookShop”/“BookShop-ejb”/“src”/“conf”选项，单击鼠标右键，在弹出的菜单中依次选择“New”/“File/Folder”选项，出现如图 22-18 所示窗口。

(3) 在“Categories”列表框中选中“XML”选项，则“File Types”列表框中列出了可选的文件类型。在此选中“XML Document”选项。单击“Next”按钮，出现如图 22-19 所示窗口。

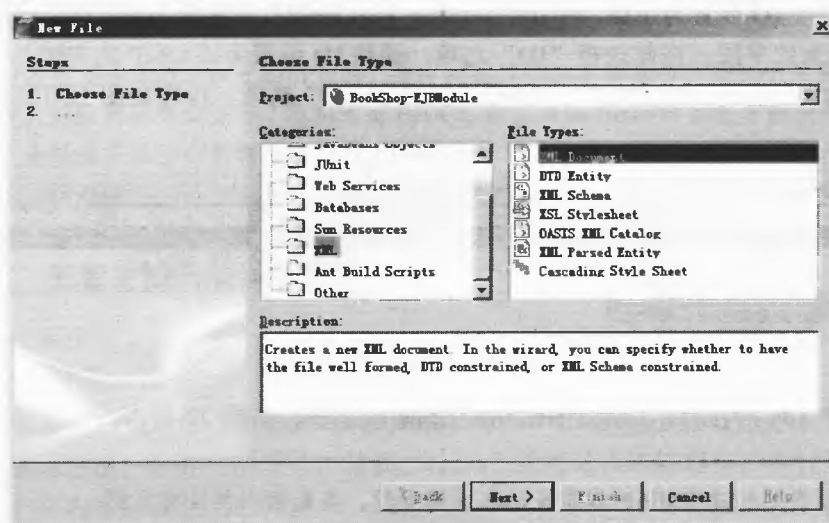


图 22-18 选择添加文件的类型

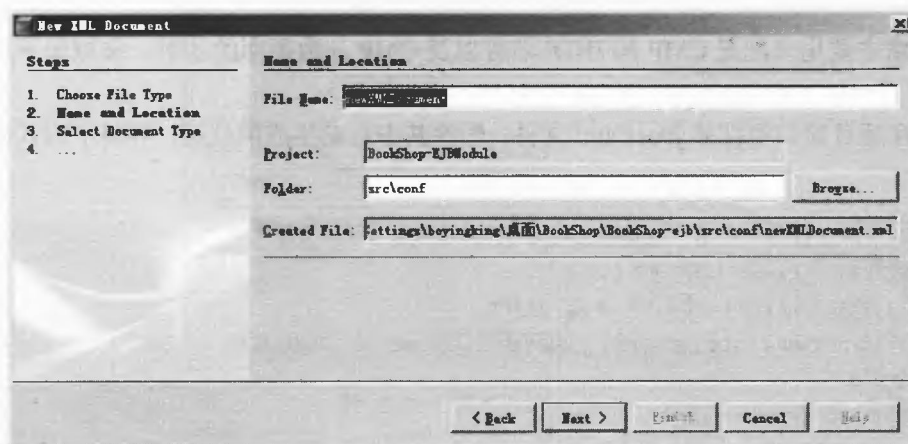


图 22-19 添加 XML 窗口

(4) 可以在如图 22-19 所示的窗口中设置要添加的 XML 文档的名称, 在此设置为 jbosscmp-jdbc。单击“Next”按钮, 此时窗口如图 22-20 所示。

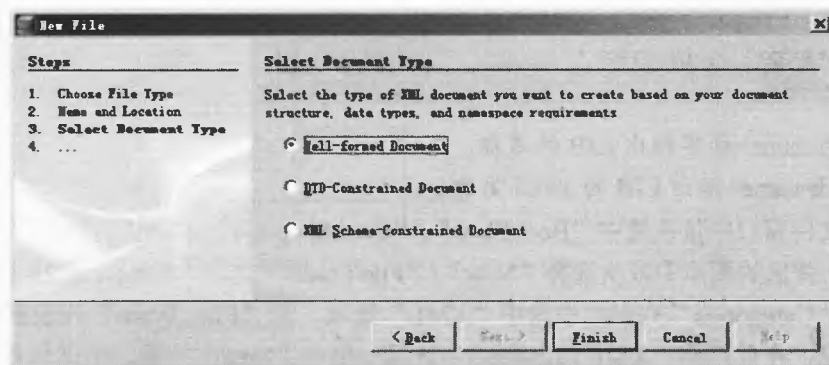


图 22-20 设置类型

(5) 可以在如图 22-20 所示的窗口中设置要添加的 XML 文件的类型, 在此使用默认的设置。单击“Finish”按钮完成。

(6) 此时在编辑器窗口中会自动打开 jbosscmp-jdbc.xml 文件。删除其中自动生成的代码, 并将下列代码添加到其中:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jbosscmp-jdbc PUBLIC "-//JBoss//DTD JBOSSCMP-JDBC 3.2//EN"
"http://www.jboss.org/j2ee/dtd/jbosscmp-jdbc_3_2.dtd">
<jbosscmp-jdbc>
  <defaults>
    <datasource>java:cmpbook</datasource>
    <datasource-mapping>Hypersonic SQL</datasource-mapping>
  </defaults>
  <enterprise-beans>
    <entity>
      <ejb-name>BookShopBean</ejb-name>
      <create-table>true</create-table>
      <remove-table>false</remove-table>
      <table-name>bookshop</table-name>
      <cmp-field>
        <field-name>name</field-name>
        <column-name>name</column-name>
      </cmp-field>
      <cmp-field>
        <field-name>price</field-name>
        <column-name>price</column-name>
      </cmp-field>
      <cmp-field>
        <field-name>id</field-name>
        <column-name>id</column-name>
      </cmp-field>
    </entity>
  </enterprise-beans>
</jbosscmp-jdbc>
```

jbosscmp-jdbc.xml 定义了 CMP 的各个选项与表之间的映射关系, 其中各个选项的具体含义如下。

- <datasource>标签定义了该 CMP 使用的数据源的名称。
- <ejb-name>标签指出了 EJB 的名称, 该名称要与 ejb.jar 文件中定义的名称相同。
- <create-table>标签指出了如果表不存在是否自动创建表。
- <remove-table>标签指出了如果表存在是否移除表。
- <table-name>标签指出了表的名称。
- <cmp-field>标签定义了 CMP 中的关系字段与表中列的映射情况。其中<field-name>表示关系字段的名称, <column-name>表示列的名称。

编译并打包 EJB 模块, 如果成功, 则出现如图 22-21 所示的窗口。

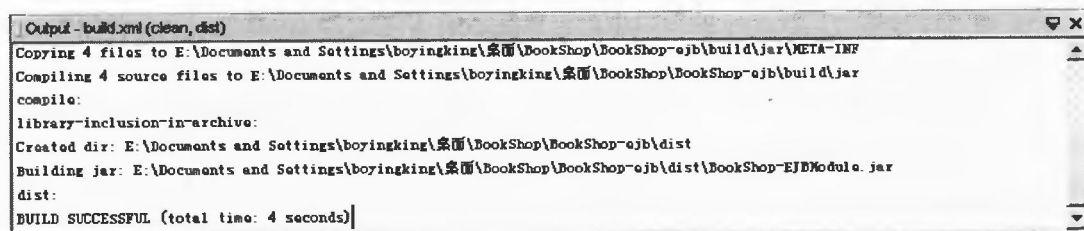


图 22-21 编译打包后输出窗口

完成了上述步骤，就已经成功地完成了 CMP 的开发与配置。

22.6 开发 Web 模块

在上一节中已经开发了负责数据库操作的 CMP 模块，在本小节中将介绍如何开发和终端用户交互的 Web 模块。Web 模块的组成部分以及功能已经在表 22-2 中给出，本节将和用户交互的 Web 模块。

22.6.1 控制器 Servlet 的开发

本小节将开发本例中用到的 Servlet，按照如下步骤完成该 Servlet 的开发。

(1) 将 BookShop-EJBModule.jar 文件添加到当前路径中，添加方法及注意事项可以参照 20.6.2 小节步骤 (16) ~ (19)。

(2) 向 BookShop-WebModule 中添加一个名称为 org.netbeans.bookshop 的包。

(3) 向 org.netbeans.bookshop 包中添加一个名称为 BookShopServlet，URL 映射值为 /BookShopServlet 的 Servlet。

(4) 向 BookShopServlet 中添加一个名称为 remoteHome，类型为 BookShopRemoteHome，初始值为 null 的成员变量。

(5) 向 BookShopServlet 中添加一个签名为 private BookShopRemoteHome lookupBookShopBeanIndiName() 的方法，并将下列代码添加到该方法中：

```
BookShopRemoteHome rv=null;
try {
    Hashtable ht = new Hashtable();
    ht.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
    ht.put(Context.PROVIDER_URL, "localhost:1099");
    Context ctx = new InitialContext(ht);
    Object remote = ctx.lookup("BookShopBeanIndiName");
    rv = (BookShopRemoteHome)
        javax.rmi.PortableRemoteObject.narrow(remote, BookShopRemoteHome.class);
} catch (javax.naming.NamingException ne) {
    ne.printStackTrace();
}
return rv;
```


- 该方法的主要功能是获取并返回“BookShopBean” EJB 的 Home 句柄。
- `ctx.lookup("BookShopBeanJndiName")` 通过前面在 `jboss.xml` 文件中配置的 EJB JNDI 名称来获取远程对象引用。

(6) 向 `BookShopServlet` 中添加一个签名为 `public void init()throws ServletException` 的方法, 并将下列代码添加到该方法中:

```
this.remoteHome=this.lookupBookShopBeanJndiName();
```

- 该方法重写了父类中的方法。
- 使用 `lookupBookShopBeanJndiName` 方法的返回值初始化 `remoteHome` 成员变量。

(7) 向 `BookShopServlet` 中添加一个签名为 `private void forward(HttpServletRequest request, HttpServletResponse response, String url) throws ServletException, IOException` 的方法, 该方法主要用来将页面跳转到由参数 `url` 指定的页面。将下列代码添加到该方法中:

```
RequestDispatcher dispatcher = request.getRequestDispatcher(url);
dispatcher.forward(request, response);
```

(8) 向 `BookShopServlet` 中添加一个签名为 `public void showAllBook(HttpSession session, HttpServletRequest request, HttpServletResponse response)` 的方法, 并将下列代码添加到该方法中:

```
try {
    //调用 EJB 中的远程方法, 其返回值为 Collection
    Collection allBook = remoteHome.findAll();
    Iterator ite=allBook.iterator();
    Vector vec=new Vector();
    while(ite.hasNext()){//遍历该 Iterator
        Object obj=ite.next();
        //将 obj 转换为 BookShopRemote 对象
        BookShopRemote temp=(BookShopRemote)
            PortableRemoteObject.narrow(obj, BookShopRemote.class);
        String tempId=new String(temp.getId().getBytes("ISO-8859-1"),
            "GB2312");//获取 id 值
        //获取 name 值
        String tempName=new String(temp.getName().getBytes("ISO-8859-1"),
            "GB2312");
        //获取 price 的值
        String tempPrice=new String
            (temp.getPrice().toString().getBytes("ISO-8859-1"), "GB2312");
        vec.add(tempId);//将 id 值添加到 Vector 中
        vec.add(tempName);//将 name 值添加到 Vector 中
        vec.add(tempPrice); //将 price 值添加到 Vector 中
    }
    session.setAttribute("all", vec);//将 vec 对象添加到名称为 all 的 session
    //对象中
    this.forward(request, response, "showadddelete.jsp");//转到
```

```
//showadddelete.jsp 页面
} catch (ServletException ex) {
    ex.printStackTrace();
} catch (RemoteException ex) {
    ex.printStackTrace();
} catch (UnsupportedEncodingException ex) {
    ex.printStackTrace();
} catch (FinderException ex) {
    ex.printStackTrace();
} catch (IOException ex) {
    ex.printStackTrace();
}
}
```

- 该方法用来获取数据库中的记录，并将这些记录的相应值添加到名称为 vec 的 Vector 对象中。
- 将 vec 添加到名称为 all 的 session 对象中。

(9) 删除 processRequest 方法体内自动生成的代码，并将下列代码添加到该方法中：

```
response.setContentType("text/html;charset=gb2312");
request.setCharacterEncoding("gb2312");
HttpSession session=request.getSession(true);
//获取 action 的值，根据 action 值的不同，进行相应的操作
String action=request.getParameter("action").trim();
String id=request.getParameter("id");//获取 ID 的值
String name=request.getParameter("name");//获取 name 的值
String price=request.getParameter("price");//获取 price 的值
try {
    //如果 action 的值为添加或 add，则向数据库中添加数据
    if(action.equals("addbook")){
        if(id!=null&&!id.trim().equals("")){//id 值不为空
            //调用 create 方法
            remoteHome.create(new String(id.getBytes(),"ISO-8859-1"),
                new String(name.getBytes(),"ISO-8859-1"),Double.valueOf(price));
            session.setAttribute("add","ok");//将 add 属性的值设置为 ok
        }
        this.forward(request,response,"addbook.jsp");//转到 addbook
        //.jsp 页面
    }else if(action.equals("delete")){
        //如果 action 属性的值为 delete，则删除指定 ID 的记录
        BookShopRemote remote=remoteHome.findByPrimaryKey(id);
        if(remote!=null){
            remote.remove();
            this.showAllBook(session,request,response);
        }
    }else if(action.equals("change")){
        //如果 action 的值为更改或 delete，则删除指定 ID 的记录
        if(id!=null&&!id.trim().equals("")){
```

```

        BookShopRemote remote=remoteHome.findByPrimaryKey(id);
        if(remote!=null){
            remote.setName(new String(name.getBytes(),"ISO-8859-1"));
            remote.setPrice(Double.valueOf(price));
            session.setAttribute("change","ok");//将 change 属性的值设置为 ok
        }else{//删除不成功
            session.setAttribute("change","no");//将 change 属性的值设置为 no
        }
    }
    this.forward(request,response,"changebook.jsp");//转到 changebook.jsp 页面
} else if(action.equals("showall")){
    //如果 action 的值为 showall, 则获取数据库中的所有记录
    this.showAllBook(session,request,response);
}
} catch (javax.ejb.RemoveException ex) {
    ex.printStackTrace();
} catch (javax.ejb.FinderException ex) {
    ex.printStackTrace();
} catch (RemoteException ex) {
    ex.printStackTrace();
} catch (UnsupportedEncodingException ex) {
    ex.printStackTrace();
} catch (CreateException ex) {
    ex.printStackTrace();
}
}

```

完成了上述步骤, 就已经完成了 Servlet 的开发。

22.6.2 开发 JSP

本项目中一共用到了 3 个 JSP 页面, 按照如下步骤完成 JSP 页面的开发。

- (1) 向 BookShop-WebModule 中添加一个名称为 addbook.jsp 的 JSP 文件。
- (2) 在编辑器窗口中打开该文件, 删除其中自动生成的代码, 并将下列代码添加到其中:

```

<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
        <title>添加图书</title>
    </head>
    <body> <h4>
        <h3>请输入要添加图书的信息: </h3>
        <form action="BookShopServlet" method="post">

```

```

        图书编号: <input type="text" name="id"><br>
        图书名称: <input type="text" name="name"><br>
        图书价格: <input type="text" name="price"><br>
        <input type="hidden" value="addbook" name="action">
        <input type="submit" value=" 添加 " >
    </form></h4>
        <a href="BookShopServlet?action=showall">查看所有图书</a>
<%
    Object obj=session.getAttribute("add");
    if(session.getAttribute("add")!=null){
        if(((String)obj).equals("ok")){
            out.println("<hr>添加成功");
        }else{
            out.println("<hr>添加失败");
        }
    }
    session.setAttribute("add",null);
%>
</body>
</html>

```

(3) 向 BookShop-WebModule 中添加一个名称为 changebook.jsp 的 JSP 文件。

(4) 删除 changebook.jsp 中自动生成的代码, 并将下列代码添加到其中:

```

<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
        <title>更改图书信息</title>
    </head>
    <body><h4>
        <form action="BookShopServlet" method="post">
            要更改信息的图书编号: <input type="text" name="id"><hr>
            请输入新内容: <br>
            图书名称: <input type="text" name="name"><br>
            图书价格: <input type="text" name="price"><br>
            <input type="hidden" value="change" name="action">
            <input type="submit" value=" 更改 " >
        </form></h4>
        <a href="BookShopServlet?action=showall">查看商品信息</a>
    <%
        Object obj=session.getAttribute("change");
        if(session.getAttribute("change")!=null){
            if(((String)obj).equals("ok")){
                out.println("<hr>更改成功");
            }else{

```

```

        out.println("<hr>更改失败");
    }
}
session.setAttribute("change", null);
%>
</body>
</html>

```

(5) 向 BookShop-WebModule 中添加一个名称为 showanddelte.jsp 的 JSP 文件。

(6) 删除 showanddelte.jsp 中自动生成的代码, 并将下列代码添加到其中:

```

<%@page contentType="text/html"%>
<%@page pageEncoding="gb2312"%>
<%@page import="java.sql.*"%>
<%@page import="java.util.*"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>全部图书</title>
</head>
<table align="center" width="100%" bgcolor="dddddd" border="1"
cellpadding="0" cellspacing="0">
<tr>
<td>图书编号</td>
<td>图书名称</td>
<td>图书价格</td>
<td>&nbsp;</td>
</tr>
<%
Vector vec=(Vector)session.getAttribute("all");
if(vec==null){
    out.println("数据库中没有商品");
}else{
    for(int i=0;i<vec.size();i++){
        String tempId=(String)vec.elementAt(i++);
%>
<tr>
<td><%=tempId%></td>
<td><%= (String)vec.elementAt(i++) %></td>
<td><%= (String)vec.elementAt(i) %></td>
<td><a href="BookShopServlet?id=<%=tempId%>&action=delete">
删除</a></td>
</tr>
<%
}}
%>
</table>

```


第 23 章

消息驱动 Bean——商品问题反馈系统

前面几章介绍了几种 EJB，包括有状态会话 Bean，无状态会话 Bean 以及实体 Bean。本章将介绍消息驱动 Bean。消息驱动 Bean 是在 EJB 2.0 中引入的，其可以对来自 Java Message Service (JMS, Java 消息服务) 提供者的异步消息处理提供支持。本章主要包括如下内容：

- JMS 简介；
- 基于 JMS 的消息驱动 Bean；
- 开发简单的使用消息驱动 Bean 的程序；
- 在 NetBeans 中开发消息驱动 Bean。

23.1 Java 消息服务

本节将对 Java 消息服务 (JMS) 进行简单的介绍，主要包括如下内容：

- JMS 简介；
- JMS 的异步性；
- JMS 的消息模型。

23.1.1 JMS 简介

JMS 是一个与开发商无关的跨平台的 API，可以用于访问企业消息系统。企业消息系统（又称为面向消息的中间件）有助于通过网络在软件之间交换消息。

JMS 的角色与 JDBC 并无不同。JDBC 为访问各种不同的关系数据库提供了一个通用 API；JMS 则支持以独立于开发商的方式访问企业消息系统。

使用 JMS 的 Java 应用称为 JMS 客户 (JMS Client)，处理消息路由和存储/转发消息的系统则称为 JMS 提供者 (JMS Provider)。JMS 应用 (JMS Application) 是一个业务系统，由多个 JMS 客户以及一个 JMS 提供者（通常是一个）组成。发送消息的 JMS 客户称为生产者 (producer)，而接收消息的 JMS 客户则称为消费者 (consumer)。JMS 客户可以既是一个生产者又是一个消费者。

23.1.2 JMS 的异步性

JMS 消息机制的一个主要优点是其异步性。换句话说，一个 JMS 客户可以发送一条消息，而无须等待回应就可以继续工作。与这种灵活机制形成对照的是 Java RMI 的同步消息机制，客户每次调用一个 Bean 的方法时，都将阻塞当前线程，直至该方法完成执行为止。这种同步处理使得客户要依赖于 EJB 服务器的可用性，从而导致客户和 EJB 之间的一种紧密耦合。

JMS 客户异步地向一个目标发送消息，其他 JMS 客户也可以接收此消息。当 JMS 客户发送一个消息时，并不会等待应答，而只是向一个消息代理发送消息，此消息代理负责将消息转发给其他客户。如果一个或者多个接收者不可用，对于客户也没有任何影响；客户会继续完成自己的工作。发送消息的客户和接收消息的客户不存在耦合；发送者不依赖于接收者的可用性。

23.1.3 JMS 消息模型

JMS 主要提供了两种消息模型，即发布——定购（Publis-and-Subscribe）和点到点（Point-to-Point）。JMS 规范将这两种消息模型称为消息域（message domain）。在 JMS 术语中，发布——定购和点到点分别简写为 pub/sub 和 p2p。本章将主要介绍点到点消息模型，对于发布——定购模型的应用，读者可以查阅相关的资料。在最简单的情况下，点到点用于消息的一对一发送，如图 23-1 所示。

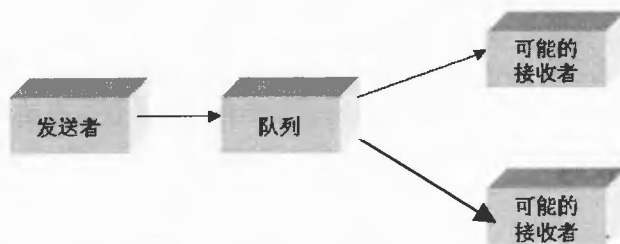


图 23-1 点到点消息模型

点到点消息模型允许 JMS 客户通过队列，以同步或者异步的方式发送和接收消息。点到点消息模型是一种基于拉（pull-based）或基于轮询（polling-based）的模型，即客户要从队列中请求消息，而不是将消息自动推至客户。

一个消息队列可能有多个接收者，但对于每条消息则只能有一个接收者。JMS 提供者要负责将消息在 JMS 客户中分发，以确保每条消息仅由一个 JMS 客户消费，如图 23-1 所示。JMS 规范并没有指出在多个接收者中分发消息的规则。

23.2 基于 JMS 的消息驱动 Bean

消息驱动 Bean（Message-driven bean，MDB）是无状态的服务器端事务性组件，用于处理通过 JMS 发送的异步 JMS 消息。消息驱动 Bean 要负责处理消息，其容器则负责管理组件的环境，包括事务、安全性、资源、并发性和消息确认。

消息驱动 Bean 类似于一个事件的接收者，只是其接收的是消息，而不是事件。正在等

待消息服务的客户，不需要等待其所请求的服务完成。消息一旦成功发送到 JMS 消息队列或主题中，这个调用服务发送消息的客户就可以移动到其他任务上，所请求的任务由消息驱动 Bean 异步完成。图 23-2 显示了消息驱动 Bean、消息服务器、消息队列以及发送消息的客户端之间的关系。

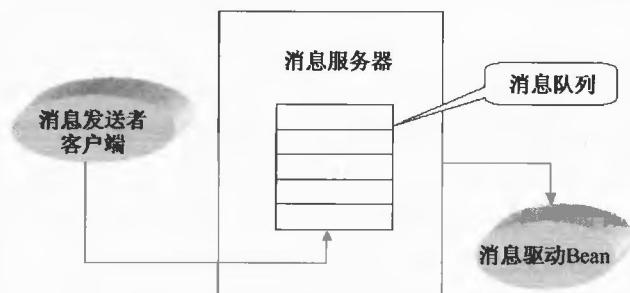


图 23-2 消息服务各个组件关系图

从图中可以看到，发送者把消息发送到相应的消息队列，消息服务器接收到消息后，调用相应的消息驱动 Bean，来完成由消息内容指定的任务。

消息驱动 Bean 是一个完整的 EJB，这与会话 Bean 和实体 Bean 是相似的。除此之外，消息驱动 Bean 主要具有如下特点。

- 消息驱动 Bean 只是服务器端组件，因此不像其他 Bean，其没有 Home 与 Remote 接口。
- 消息驱动 Bean 必须实现 MessageDrivenBean 与 MessageListener 两个接口。
- 消息驱动 Bean 有且只有一个名称为 onMessage 的业务方法，该方法在接收消息时，由容器调用。
- 每个已经部署的消息驱动 Bean 与部署描述符中指定的 JMS 目标所关联，而其他的 EJB 可以接收任何客户端的调用。
- 由于其异步特性，因此消息驱动 Bean 不直接返回值，也不抛出异常。因为所谓调用消息驱动 Bean 只是向消息服务器发送消息，而实际调用 onMessage 方法的是消息驱动 Bean 容器。

23.3 开发一个简单的使用消息驱动 Bean 的程序

上面两节对 JMS 和消息驱动 Bean 进行了简单的介绍，本节将开发一个简单的使用消息驱动 Bean 的程序。主要包括如下内容：

- 配置消息服务；
- 开发、配置与部署消息驱动 Bean；
- 开发与配置控制器 Servlet；
- 开发 JSP；
- 运行项目。

23.3.1 项目功能简介

本节开发了一个使用消息驱动 Bean 的程序，其功能比较简单，具体功能如下。

- 用户可以在客户端中输入两个参数，在 Servlet 中接收并处理这两个参数，并把参数封装到 JMS 消息中发送到消息队列。
- 在消息驱动 Bean 中获取用户所提交的参数，并在 JBoss 控制台打印这些内容。

23.3.2 配置消息服务

在开发消息驱动 Bean 之前，首先要配置消息服务，一般是在消息服务中增加一个消息队列或主题。这样消息的发送者才可以向指定的消息队列或主题发送消息，同时消息驱动 Bean 也需要注册到指定的消息队列或主题。

由于 JBoss 是一个开源的应用服务器，所以不像 WebLogic 具有友好的用户配置界面，在 JBoss 中配置消息服务的方式就是编辑相应的 XML 配置文件。配置消息服务主要是配置 JBoss4 安装目录 server/default/deploy/jms 下的 jbossMQ-destinations-service.xml 文件。jbossMQ-destinations-service.xml 用于定义具体应用所需的目的地（也就是消息队列或主题）信息，通过编辑它，可以配置消息队列的 JMS 应用的目的地。

在本例中，需要将如下代码添加到 jbossMQ-destinations-service.xml 中 <server> 与 </server> 标记之间：

```
<mbean code="org.jboss.mq.server.jmx.Queue"
  name="jboss.mq.destination:service=Queue,name=HelloQueue">
  <depends optional-attribute-name="DestinationManager">
    jboss.mq:service=DestinationManager</depends>
  <depends optional-attribute-name="SecurityManager">
    jboss.mq:service=SecurityManager</depends>
  <attribute name="MessageCounterHistoryDayLimit">-1</attribute>
  <attribute name="SecurityConf">
    <security>
      <role name="guest" read="true" write="true"/>
      <role name="publisher" read="true" write="true" create="false"/>
      <role name="noacc" read="false" write="false" create="false"/>
    </security>
  </attribute>
</mbean>
```

- <mbean> 标签中 code 属性的值定义了消息的类型，name 属性的值定义了消息目的地名称。如果要更改消息队列中消息目的地的名称，只需更改 name=HelloQueue 处 HelloQueue 的值即可。
- <depends optional-attribute-name="DestinationManager"> 定义了消息服务器中管理目的地服务的对象名称。
- <depends optional-attribute-name="SecurityManager"> 定义了用于验证客户请求的安全性管理服务对象的名称。
- <attribute name="MessageCounterHistoryDayLimit"> 计算目的地消息的历史天数，如果小于 0，则不限制天数；如果等于 0，则不给出历史天数；如果大于 0，则计算该大小的历史天数。
- <security> 标签用于定义相应角色的用户所具有的权限。具体的角色及其权限由

完成了上述步骤，就完成了消息服务的配置。启动 JBoss，如果出现如图 23-3 所示的内容，则表明配置成功。

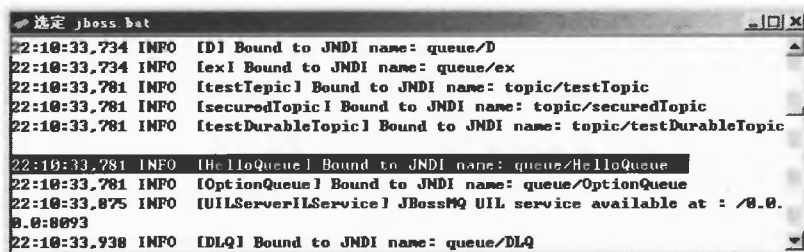


图 23-3 成功配置消息服务

提示 消息服务的 JNDI 名称为“queue/目的地名称”。如前面配置的消息目的地名称为 HelloQueue, 则消息服务的 JNDI 名称为 queue/HelloQueue。

本节中创建了一个名称为 `HelloMDBean` 的消息驱动 Bean，该消息驱动 Bean 用于接收消息，获取消息的内容并打印出来。下面给出了该消息驱动 Bean 的完整代码：

```
package org.netbeans.jms;
import java.util.*;
import javax.ejb.*;
import javax.jms.*;
import java.rmi.*;
import javax.sql.*;
import javax.naming.*;

public class HelloMDBean implements MessageDrivenBean, MessageListener {
    private MessageDrivenContext context;
    public void ejbCreate() { //do nothing
    }
    public void ejbRemove() { //do nothing
    }
    public void setMessageDrivenContext(MessageDrivenContext context) {
        this.context = context; //初始化上下文
    }
    public void onMessage(Message message) {
        //将接收的 Message 类型参数转换为 MapMessage 类型
        MapMessage mapmessage = (MapMessage) message;
        try {
            Enumeration e = mapmessage.getMapNames();
            String[] val=new String[2];
            int flag=0;
            System.out.println("=====以下是从客户端获取的内容=====");
            System.out.println("第一个参数\t\t\t第二个参数");
```

```

        while (e.hasMoreElements()) {
            String key = (String) e.nextElement(); // 获取键
            val[flag] = mapmessage.getString(key); // 获取值, 并将该值赋值与数组
            flag++;
        }
        System.out.print(val[0]+"\\t\\t"+val[1]); // 输出数组内容
        System.out.println("=====");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
}

```

完成代码开发以后, 还要对该 Bean 进行相应的配置。操作步骤如下。

(1) 在当前系统中创建一个名称为 MDBean 的目录, 并在该目录中新建一个名称为 mdb 的子目录, 将编译后的 Java 类文件 (包括包) 添加到该目录 (mdb) 中。

(2) 在 mdb 目录中新建一个名称为 META-INF 的目录, 并向其中分别添加名称为 ejb-jar.xml 与 jboss.xml 的两个文件。

(3) 将以下代码添加到 ejb-jar.xml 文件中:

```

<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>HelloMDBean</ejb-name>
      <ejb-class>org.netbeans.jms>HelloMDBean</ejb-class>
      <transaction-type>Container</transaction-type>
      <message-driven-destination>
        <destination-type>javax.jms.Queue</destination-type>
      </message-driven-destination>
    </message-driven>
  </enterprise-beans>
</ejb-jar>

```

(4) ejb-jar.xml 配置了与 EJB 有关的各个选项的值。该例中各标签的含义如下。

- <ejb-jar>文档的根, 所有的其他元素都必须位于此元素内。
- <enterprise-beans>XML 中声明的每一个 Bean 都必须位于此元素内。
- <message-driven>标签用来说明当前配置的 EJB 为消息驱动 Bean。
- <ejb-class>标签用来定义消息驱动 Bean 的类, 值为该类的全称类名。
- <transaction-type>用来定义持久性类型。
- <message-driven-destination>用来定义消息的目的地。<destination-type>为 javax.jms.Queue, 说明消息驱动 Bean 通过一个点到点的消息模型, 从一个消息队列中获取消息进行消费。

(5) 将以下代码添加到 jboss.xml 文件中:


```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC
    "-//JBoss//DTD JBOSS 4.0//EN"
    "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss>
  <enterprise-beans>
    <message-driven>
      <ejb-name>HelloMDBean</ejb-name>
      <configuration-name>Standard Message Driven Bean</configuration-name>
      <destination-jndi-name>queue/HelloQueue</destination-jndi-name>
    </message-driven>
  </enterprise-beans>
</jboss>

```

- ① <ejb-name>指定消息驱动 Bean 的 EJB 名称, 该名称与 ejb-jar.xml 文件中<ejb-name>元素的值相同。
- ② <configuration-name>指定配置的名称, 一般在 IDE 中配置消息驱动 Bean 时使用。
- ③ <destination-jndi-name>指定目的地的 JNDI 名称, 该名称与 23.3.1 中配置的消息服务的 JNDI 名称要一致。

(6) 此时 mdb 的目录组织结构如图 23-4 所示。

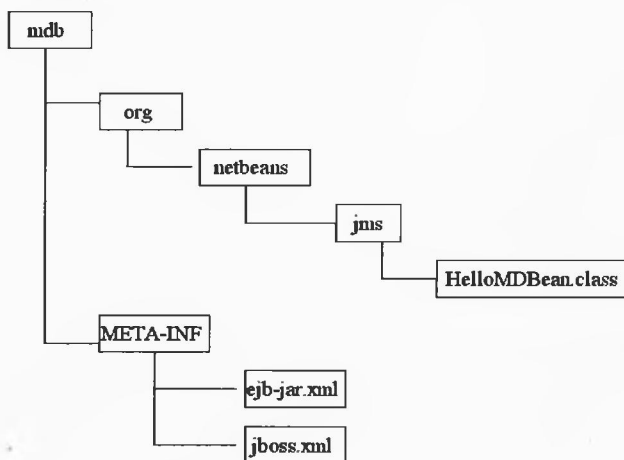


图 23-4 mdb 的目录组织结构图

(7) 在 MDBean 目录中, 新建一个名称为 mdb.bat 的文件, 该文件用来将消息驱动 Bean 及其配置文件打包。用记事本打开该文件, 将以下代码添加到其中:

```

cd bean
jar cvf HelloMDBean.jar *

```



提示

如果发现 jar 命令不可用, 则需要把 JDK 安装目录下的 bin 目录设置到 PATH 环境变量中, 具体操作过程可以参照本书 19.7.1 步骤 (4) 中有关 PATH 环境变量的部分。

(8) 双击运行 mdb.bat, 此时在 bean 目录中会出现一个名为 HelloMDBean.jar 的文件。

(9) 启动 JBoss, 将该文件拷贝到 JBoss 安装目录下的 server/default/deploy 中。如果出现如图 23-5 所示界面的最后三行, 则说明部署成功。

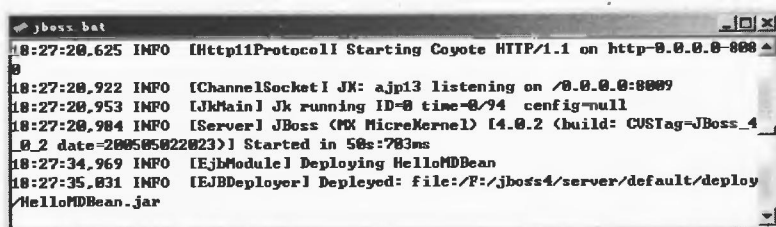


图 23-5 成功部署消息驱动 Bean 界面

23.3.4 开发与配置控制器 Servlet

上个小节中完成了消息驱动 Bean 的开发配置与部署, 本小节将完成 Web 客户端的开发配置以及部署。本例中开发了一个名称为 HelloMDBeanServlet 的 Servlet, 该 Servlet 接收一些参数, 并将收到的参数值包装成 MapMessage 对象后发送。下面给出了该 Servlet 的完整代码:

```
package org.netbeans.jms;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import javax.ejb.*;
import javax.rmi.*;
import javax.naming.*;
import java.util.*;
import java.rmi.*;
import javax.jms.*;
import javax.naming.*;
import javax.jms.Queue;
public class HelloMDBeanServlet extends HttpServlet {
    private static String QUEUE_NAME="queue/HelloQueue";
    private static String CONNECTION_FACTORY = "ConnectionFactory";
    private static String PROVIDER_URL="localhost:1099";
    private QueueSender sender;// 声明 JMS 发送者
    private QueueSession session;//声明 JMS session 对象
    public void doPost(HttpServletRequest req,HttpServletResponse res)
    throws IOException {
        res.setContentType("text/html;charset=GBK");
        req.setCharacterEncoding("GBK");
        String firstParam=req.getParameter("firstParam");//获取参数值
        String secondParam=req.getParameter("secondParam");
        Hashtable msg = new Hashtable();
        msg.put("firstParam",firstParam);
        msg.put("secondParam",secondParam);
        PrintWriter pw=res.getWriter();
```

```

try {
    Context ctx = getInitialContext();
    // 查找 JMS 工厂
    QueueConnectionFactory conFactory =
        (QueueConnectionFactory) ctx.lookup(CONNECTION_FACTORY);
    // 创建 JMS 连接
    QueueConnection connection = conFactory.createQueueConnection();
    // 创建 JMS 会话
    session = connection.createQueueSession
        (false, QueueSession.AUTO_ACKNOWLEDGE);
    // 查找 JMS 队列
    Queue chatQueue = (Queue) ctx.lookup(QUEUE_NAME);
    // 创建 JMS 发送者
    sender = session.createSender(chatQueue);
    // 发送消息
    this.sendMsg(msg);
    pw.println("<h3>发送消息成功<h3>");
} catch (Exception e) {
    e.printStackTrace();
    pw.println("发送消息失败<br>");
    pw.println(e);
}
pw.println("<br><a href=sendmessage.jsp>返回</a>");
}
//获取初始化上下文
private Context getInitialContext() throws NamingException{
    Properties env = new Properties();
    env.put("java.naming.factory.initial", "org.jnp.interfaces.NamingContextFactory");
    env.put("java.naming.provider.url", PROVIDER_URL);
    return new InitialContext(env);
}
//该方法用于将 Hashtable 转换为 MapMessage 对象，并调用 send 方法发送消息
public void sendMsg(Map msg) throws Exception {
    MapMessage message = session.createMapMessage();
    Set keys = msg.keySet(); //获取消息的键
    Iterator i = keys.iterator();
    while (i.hasNext()) {
        String key = (String) i.next();
        String val = (String) msg.get(key);
        message.setString(key, val); //将对应的键-值对添加到 MapMessage 中
    }
    // 将 message 发送到消息队列
    sender.send(message);
}
}

```

从以上代码可以看到, 使用 Servlet 发送消息, 并不是简单的工作。要想能够发送消息, 需要进行如下工作。

- 获取环境上下文。
- 查找 JMS 工厂。
- 使用 JMS 工厂创建 JMS 连接。
- 使用 JMS 连接创建 JMS 会话。
- 查找 JMS 队列。
- 创建 JMS 消息发送者。
- 把要发送的信息封装进消息对象。
- 发送消息。

提示

以上代码首先将客户端获取的数据添加到 Hashtable 中, 然后在 sendMsg 方法中, 将 Hashtable 中的数据转换到 MapMessage 中。这样看似麻烦, 实则不然, 因为在实际的工作中, 这两个工作可能会由不同的人员完成, 普通的开发人员使用 Hashtable, 减少了开发难度。

开发完 Servlet 以后, 要进行相应的配置, 操作步骤如下。

(1) 在前面创建的 MDBean 目录中, 创建一个名称为 war 的子目录, 在 war 目录中创建 WEB-INF\classes 目录, 将编译后的 HelloMDBeanServlet 类文件添加到 classes 目录中。

(2) 在 WEB-INF 目录中新建一个名称为 web.xml 的 xml 文件, 并将下列代码添加到其中。该配置文件主要用来配置 Servlet。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<display-name>DefaultApp</display-name>
  <servlet>
    <servlet-name>HelloMDBeanServlet</servlet-name>
    <servlet-class>org.netbeans.jms.HelloMDBeanServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloMDBeanServlet</servlet-name>
    <url-pattern>/HelloMDBeanServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

完成了上述步骤, 就已经成功地完成了 Servlet 的开发与配置。

23.3.5 开发 JSP

本小节开发了一个名称为 sendmessage 的 JSP 文件, 其代码如下:

```
<%@ page contentType="text/html; charset=GBK"%>
<html>
```

```

<head>
  <title>消息驱动 Bean</title>
</head>
<body><h3>
  <form action="HelloMDBeanServlet" method="post">
    第一个参数: <input type="text" size="20" name="firstParam"><br>
    第二个参数: <input type="text" size="20" name="secondParam"><br>
    <input type="submit" value="提交">
  </form>
</h3></body>
</html>

```

- 该 JSP 将数据提交到 HelloMDBeanServlet。
- 将该 JSP 文件保存在 war 目录中。

将以下代码添加到 web.xml 文件中,用来将 sendmessage.jsp 设置为该 Web 应用的欢迎界面。

```

<welcome-file-list>
  <welcome-file>sendmessage.jsp</welcome-file>
</welcome-file-list>

```

此时 war 的目录组织结构如图 23-6 所示。

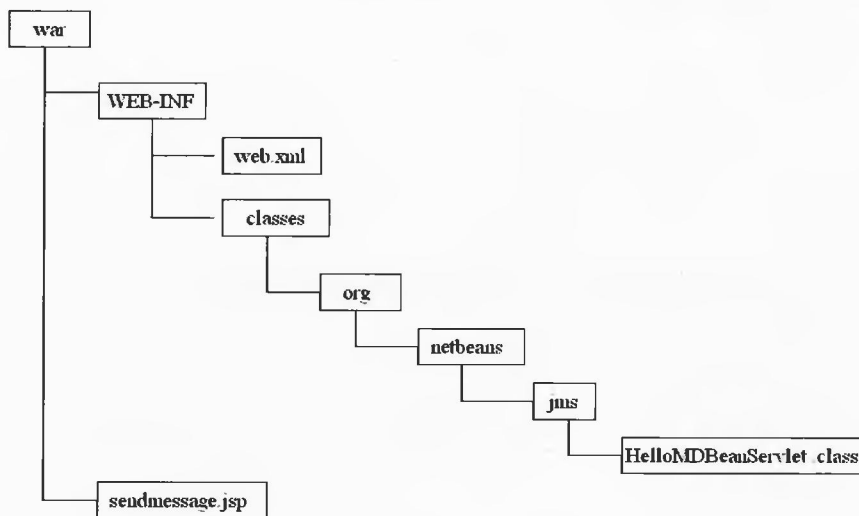


图 23-6 war 目录的组织结构图

在 MDBean 目录中,新建一个名称为 war.bat 的 bat 文件,该文件用来将 Servlet、JSP 文件及其配置文件压缩为 war 包。用记事本打开该文件,将下列内容添加到其中:

```

cd war
jar cvf Hello.war *

```

双击运行 war.bat 文件,运行完毕后,在 war 目录中将出现一个名称为 Hello.war 的文件。启动 JBoss,将该文件拷贝到 JBoss 安装目录下 server/default/deploy 目录中。如果出现如图 23-7 所示界面最后两行的消息,则说明部署成功。

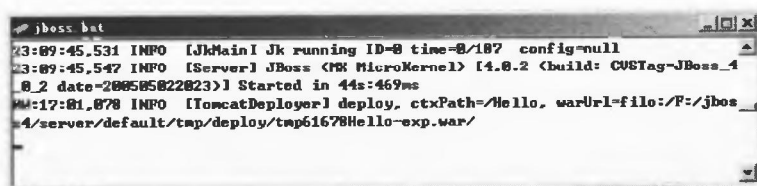


图 23-7 部署 Web 程序

23.3.6 运行项目

完成了项目的开发以及部署以后，就可以运行项目了。启动 JBoss 以后，在浏览器的地址栏中输入如下内容：

`http://localhost:8080/Hello/`

此时浏览器中内容如图 23-8 所示。输入一定内容后单击“提交”按钮，此时浏览器中内容如图 23-9 所示。

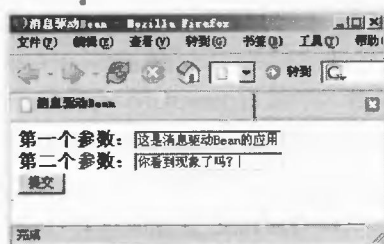


图 23-8 程序运行界面

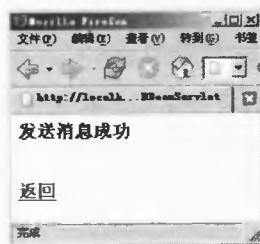


图 23-9 提交消息后界面

此时查看 JBoss 的控制台，界面如图 23-10 所示，这说明消息驱动 Bean 已经对通过 Web 发送的消息进行了处理。

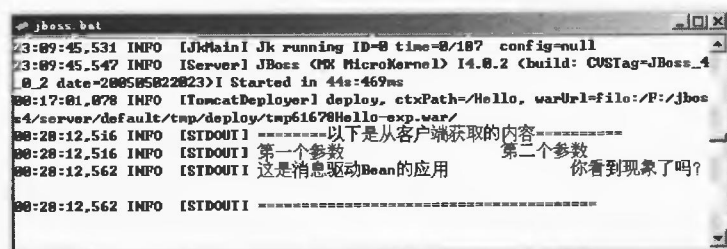


图 23-10 输出消息内容

23.4 在 NetBeans 中开发消息驱动 Bean

在 NetBeans 中开发消息驱动 Bean 非常方便，本节将在 NetBeans 中开发一个使用消息驱动 Bean 的项目。

23.4.1 项目功能简介

本节开发了一个名称为 MDBeanExample 的项目。该项目可供用户反馈商品的一些信息，

消息驱动 Bean 负责将这些信息添加到数据库中。运行该项目，界面如图 23-11 所示。

用户可以在如图 23-11 所示的窗口中设置问题产品的名称、留言的主题、留言人的姓名、联系方式以及留言内容等。设置完毕后，单击“提交”按钮。此时界面如图 23-12 所示。

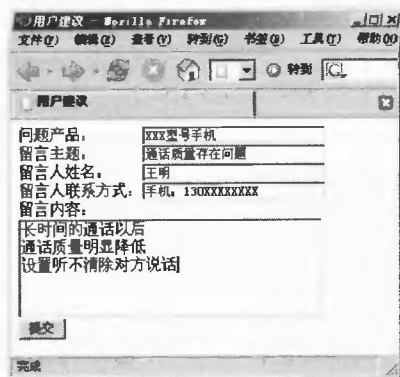


图 23-11 程序运行界面

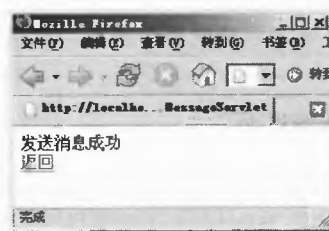


图 23-12 发送消息成功界面

此时查看数据库，可以看到相应的记录被添加到数据库中。

23.4.2 准备工作

在开发项目之前，需要进行一些准备工作，包括配置消息服务、创建数据库表、配置数据源等内容。

1. 配置消息服务

要使用消息驱动 Bean，首先要配置消息服务。将以下代码添加到 jbossMQ-destinations-service.xml 中<server>与</server>标记之间：

```
<mbean code="org.jboss.mq.server.jmx.Queue"
name="jboss.mq.destination:service=Queue,name=OptionQueue">
  <depends optional-attribute-name="DestinationManager">
    jboss.mq:service=DestinationManager</depends>
  <depends optional-attribute-name="SecurityManager">
    jboss.mq:service=SecurityManager</depends>
  <attribute name="MessageCounterHistoryDayLimit">-1</attribute>
  <attribute name="SecurityConf">
    <security>
      <role name="guest" read="true" write="true"/>
      <role name="publisher" read="true" write="true" create="false"/>
      <role name="noacc" read="false" write="false" create="false"/>
    </security>
  </attribute>
</mbean>
```

- 以上代码配置了一个消息目的地名称为 OptionQueue 的消息队列。

2. 创建数据库表

本项目使用 MySQL 作为项目的后台数据库，在数据库中创建一个名称为 options 的表，

表中各个字段的名称和类型如表 23-1 所示。

表 23-1 options 表的各个字段

字段名称	字段类型	字段名称	字段类型
bugproduct	VARCHAR(30)	title	VARCHAR(40)
name	varchar(20)	contacttype	varchar(50)
content	varchar(800)	date	varchar(40)

提示 具体的创建过程读者可以参照 19.4.4 的内容，这里不再赘述。

3. 配置数据源

在 JBoss 安装目录的 server\default\deploy 目录中新建一个名称为 mysql-wyfqueue-ds 的 xml 文件，并将下列代码添加到该文件中：

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>mdBeanDB</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/test</connection-url>
    <driver-class>org.gjt.mm.mysql.Driver</driver-class>
    <user-name>root</user-name>
    <password></password>
    <min-pool-size>5</min-pool-size>
    <max-pool-size>20</max-pool-size>
    <idle-timeout-minutes>0</idle-timeout-minutes>
    <track-statements/>
  </local-tx-datasource>
</datasources>
```

以上代码配置了一个名称为 mdBeanDB 的数据源。

23.4.3 开发与配置消息驱动 Bean

本项目中，消息驱动 Bean 主要负责从消息中获取数据，并将其添加到数据库中。本节将开发消息驱动 Bean，按照如下步骤完成其开发。

(1) 启动 NetBeans，创建一个名称为 MDBeanExample 的 Enterprise Application 项目。

(2) 向 MDBeanExample-EJBModule 中添加一个名称为 org.netbeans.jms 的包。

(3) 在项目窗口中，依次选择“MDBeanExample-EJBModule”/“Source Packages”/“org.netbeans.jms”节点，单击鼠标右键，在弹出的菜单中依次选择“New”/“Message-Driven Bean”选项，出现如图 23-13 所示的向导窗口。

(4) 可以在如图 23-13 所示的窗口中设置添加的消息驱动 Bean 的名称、位置以及消息模型的类型。在此将添加的消息驱动 Bean 的名称设置为 MessageBoardBean，消息模型的类型设置为“Queue”，单击“Finish”按钮完成。

(5) 此时项目窗口中内容如图 23-14 所示。

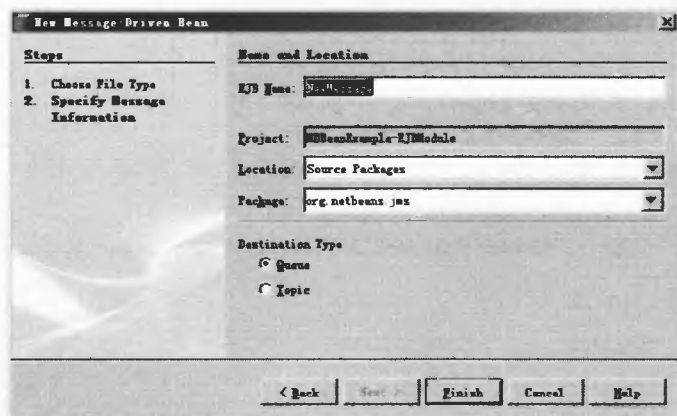


图 23-13 添加消息驱动 Bean 的向导窗口

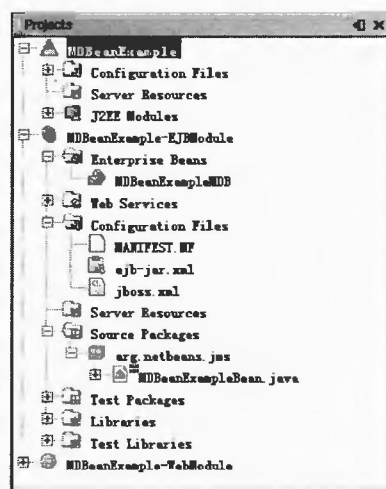


图 23-14 项目窗口中内容

(6) 向 MessageBoardBean 类中添加一个签名为 `private Connection getConnection() throws SQLException` 的方法，并将下列代码添加到该方法中：

```
Context ctx = null;
DataSource ds = null;
String url = "localhost:1099";
try {
    Hashtable h = new Hashtable( );
    h.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.Naming-
    ContextFactory");
    h.put(Context.PROVIDER_URL, url);
    ctx = new InitialContext(h);
    ds = (DataSource) ctx.lookup("java:mdBeanDB"); //查找数据源
} catch (Exception ne) {
    System.out.println("UNABLE to get a connection from !");
}
return ds.getConnection(); //返回数据库连接
```

- ① 查找 JNDI 名称为 `java:mdBeanDB` 的数据源。
- ② 返回对应于该数据源的一个数据库连接。

(7) 向 `onMessage` 方法的方法体内添加如下代码：

```
MapMessage mapmessage = (MapMessage) aMessage;
String[] val=new String[6];
try {
    //获取对应属性的值，并将其赋值于数组
    val[0]=mapmessage.getString("bugProduct");
    val[1]=mapmessage.getString("title");
    val[2]=mapmessage.getString("name");
    val[3]=mapmessage.getString("contactType");
    val[4]=mapmessage.getString("content");
```

```

        val[5]=new Date().toLocaleString();//将当前时间添加到数组中
        //将数组中值的编码转换 ISO-8859-1
        val[0]=new String(val[0].getBytes(),"ISO-8859-1");
        val[1]=new String(val[1].getBytes(),"ISO-8859-1");
        val[2]=new String(val[2].getBytes(),"ISO-8859-1");
        val[3]=new String(val[3].getBytes(),"ISO-8859-1");
        val[4]=new String(val[4].getBytes(),"ISO-8859-1");
        val[5]=new String(val[5].getBytes(),"ISO-8859-1");
        Connection con=this.getConnection();
        PreparedStatement ps=con.prepareStatement("insert into options
        values(?,?,?,?,?,?)");
        ps.setString(1,val[0]);
        ps.setString(2,val[1]);
        ps.setString(3,val[2]);
        ps.setString(4,val[3]);
        ps.setString(5,val[4]);
        ps.setString(6,val[5]);
        ps.executeUpdate();
        con.close();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

```

- 该方法用于获取消息中对应键的值。
- 将这些值转换编码以后，添加到数据库中。

(8) 选中“MDBBeanExample-EJBModule”/“Configuration Files”/“jboss.xml”节点，打开该文件，删除其中自动生成的代码，将下列代码添加到其中：

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC
    "-//JBoss//DTD JBOSS 4.0//EN"
    "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss>
    <enterprise-beans>
        <message-driven>
            <ejb-name>MessageBoardBean</ejb-name>
            <configuration-name>Standard Message Driven Bean</configuration-name>
            <destination-jndi-name>queue/OptionQueue</destination-jndi-name>
        </message-driven>
    </enterprise-beans>
</jboss>

```

完成了上述步骤，就已经成功地完成了消息驱动 Bean 的开发及配置。

23.4.4 控制器 Servlet 的开发

前面完成了消息驱动 Bean 的开发及配置，本节将完成 Servlet 的开发，按照如下步骤完成。

- (1) 向 MDBeanExample-WebModule 中添加一个名称为 org.netbeans.jms 的包。
- (2) 向 org.netbeans.jms 包中添加一个名称为 MessageServlet、URL 映射值为 /MessageServlet 的 Servlet。
- (3) 按如表 23-2 所示向 MessageServlet 中添加成员变量。

表 23-2 添加成员变量的各选项说明

变 量 名 称	变 量 类 型	访问限制修饰符号	静态的	初 始 值
QUEUE_NAME	String	private	是	"queue/OptionQueue"
CONNECTION_FACTORY	String	private	是	"ConnectionFactory"
PROVIDER_URL	String	private	是	"localhost:1099"
sender	QueueSender	private	否	null
session	QueueSession	private	否	null

- (4) 向 MessageServlet 中添加一个签名为 private Context getInitialContext() throws NamingException 的方法，该方法用于获取初始化上下文。将下列代码添加到该方法中：

```
Properties env = new Properties();
env.put("java.naming.factory.initial", "org.jnp.interfaces.NamingContextFactory");
env.put("java.naming.provider.url", PROVIDER_URL);
return new InitialContext(env);
```

- (5) 向 MessageServlet 中添加一个签名为 private void sendMsg(Map msg) throws Exception 的方法，将下列代码添加到该方法中：

```
MapMessage message = session.createMapMessage();
Set keys = msg.keySet();
Iterator i = keys.iterator();
while (i.hasNext()) {
    String key = (String) i.next();
    String val = (String) msg.get(key);
    message.setString(key, val);
}
// 将 message 发送到消息队列
sender.send(message);
```

- ① 该方法用于将 Map 类型的对象转换为 MapMessage 类型的对象。
- ② 将其发送到消息队列中。

- (6) 删除 processRequest 方法内自动生成的代码，并将下列代码添加到其中：

```
response.setContentType("text/html;charset=gb2312");
request.setCharacterEncoding("gb2312");
PrintWriter out = response.getWriter();
//获取从客户端发送的值
String bugProduct=request.getParameter("bugProduct");
```

```
String title=request.getParameter("title");
String name=request.getParameter("name");
String contactType=request.getParameter("contactType");
String content=request.getParameter("content");
Hashtable hash = new Hashtable();
//将获取的值以键-值对的形式添加到 Hashtable 对象中
hash.put("bugProduct",bugProduct);
hash.put("title",title);
hash.put("name",name);
hash.put("contactType",contactType);
hash.put("content",content);
try {
    Context ctx = getInitialContext();
    //查找 JMS 工厂
    QueueConnectionFactory conFactory =
        (QueueConnectionFactory) ctx.lookup(CONNECTION_FACTORY);
    // 创建 JMS 连接
    QueueConnection connection = conFactory.createQueueConnection();
    // 创建 JMS session 对象
    session = connection.createQueueSession(false, QueueSession.AUTO_
        ACKNOWLEDGE);
    // 查找 JMS 队列
    javax.jms.Queue chatQueue = (javax.jms.Queue) ctx.lookup(Queue_NAME);
    // 创建 JMS 发送者
    sender = session.createSender(chatQueue);
    // 发送消息
    this.sendMsg(hash); //调用 sendMsg 方法发送消息
    out.println("<h3>发送消息成功</h3>");
} catch(Exception e) {
    e.printStackTrace();
    out.println("发送消息失败<br>");
    out.println(e);
}
out.println("<br><h3><a href=options.jsp>返回</a></h3>");
out.close();
```



提示

以上代码主要用来创建一些发送消息需要用到的资源，然后调用 `sendMsg` 方法发送消息。

完成了上述步骤，就已经成功地完成了 Servlet 的开发。

23.4.5 开发 JSP

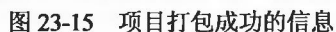
本小节将开发用于提交数据的 JSP 页面，按照如下步骤完成其开发。

- (1) 向 MDBeanExample-WebModule 中添加一个名称为 `options` 的 JSP 文件。
- (2) 在文本编辑器中打开该文件，删除其中自动生成的代码，并将以下代码添加到其中：

 **提示** action 属性的值 MessageServlet 即为此 JSP 页面的数据要提交到的 Servlet 的 URL 映射值。

23.4.6 编译并运行项目

(1) 用鼠标右键单击项目窗口中的 `MDBeanExample` 节点，选择右键菜单中的“Clean and Build Project”选项。此时系统会自动对整个项目进行编译打包，如果没有错误，输出窗口中将显示编译打包成功的信息，如图 23-15 所示。



513



23.5 小结

本章介绍了消息驱动 Bean 的开发和使用，可以体会到使用 NetBeans 开发消息驱动 Bean 是非常方便的，省去了开发人员编写很多配置文件和脚本的工作，从而有效地降低了开发成本，提高了开发效率。

本篇为告别篇，即本书的最后一篇。本篇主要对与 Java 和 NetBeans 相关的其他技术进行了简单的介绍，主要包括如下内容：

- 如何在 NetBeans 中使用已有的 Ant 脚本创建项目；
- 如何使用 JUnit 进行单元测试；
- 如何使用 NetBeans Profiler 对应用进行分析；
- 如何将已有的 Eclipse 项目导入到 NetBeans。

第 5 篇 高级工具篇

第 24 章 使用 NetBeans 集成已存在的
Ant 脚本

第 25 章 在 NetBeans 中使用 JUnit

第 26 章 NetBeans Profiler：监控应用
程序的执行

第 27 章 将 Eclipse 项目导入 NetBeans

第 24 章

使用 NetBeans 集成已存在的 Ant 脚本

本章将为读者介绍如何使用 NetBeans 集成已经存在的 Ant 脚本，主要包括如下内容：

- 在 NetBeans 中使用 Ant 脚本的简单说明；
- 创建自由格式项目；
- 为项目设置命令；
- 为文件设置命令；
- 更改自由格式项目的目标 JDK。

24.1 在 NetBeans 中使用 Ant 脚本

24.1.1 在 NetBeans 中使用 Ant 脚本的原因

NetBeans 中，标准项目的用户接口适用于通常的开发流程，可以帮助开发者培养良好的开发模式，因此特别适用于从头开发项目。

不过在标准用户接口中，并没有囊括所有开发中的情形，尤其一开始在其他开发环境下开发的项目。对于这种情况，可以利用 NetBeans 与 Ant 脚本集成的特性，定制 NetBeans，使用已经存在的 Ant 生成脚本。在下面的两种情况下，一般会在 NetBeans 中使用 Ant 脚本。

- 如果项目已经存在生成脚本，并且不需要（或者不能）使用 NetBeans 重新生成脚本，可以对 NetBeans 进行必要的配置，然后使用已经存在的脚本创建自由格式项目。
- 在标准项目的各种限制条件无法满足的情况下，NetBeans 中使用 Ant 脚本生成自由格式项目更受欢迎。

提示 在使用 NetBeans 中已经存在的生成脚本前，应当仔细考虑是否真的需要使用已经存在的 Ant 脚本，并且确定使用标准的 NetBeans 项目无法实现特定的效果，从长远来说，标准 NetBeans 项目更容易维护。

24.1.2 NetBeans 的项目系统与自由格式项目的区别

NetBeans 的项目系统与 Ant 工具紧密结合，以至于增量输出型的命令（比如编译单个文件）、以及其他需要特别与 NetBeans 应用结合的命令（如调试），都定义在生成脚本中。不过，NetBeans 的项目系统与自由格式的项目也有所区别，原因如下。

- 在标准项目中，增量输出型命令的生成目标都会由 NetBeans 默认生成。
- 在自由格式项目中，开发者可以自由编写增量输出型命令的生成目标，从而使其能够按照需要对项目进行各种操作。

24.1.3 在 NetBeans 中使用 Ant 脚本的步骤

- （1）使用自由格式（已经存在的 Ant 脚本）模版，在 NetBeans 中创建项目。
- （2）将已存在的主要生成目标，映射到 NetBeans 中对应的命令（比如“编译与运行应用程序”或“运行与调试”等）。在新建项目向导中，可以创建这些映射；或者在创建项目以后，通过项目属性对话框来创建这些映射。
- （3）在新建项目向导（创建项目时）或项目属性对话框（创建项目后）中注册类路径项（比如外部源文件根目录或者 JAR 文件），从而使 NetBeans 的某些功能可以正常工作。
- （4）为 NetBeans 中的常用命令创建新的生成目标，并在项目的 `project.xml` 文件中，创建对应这些目标的映射。对于这些命令，在第一次运行给定的命令时，NetBeansIDE 帮助生成对应的目标（以及 `project.xml` 文件中的映射）。不过有可能需要对该目标进行修改，才能使其在项目中正常工作。
- （5）对于其他命令，只能从头编写对应的目标，并且手动在项目的 `project.xml` 中创建映射。在 NetBeans 中，可以为“调试项目”和“编译项目”命令生成目标与映射，而其他命令的目标则需要开发者自己创建，比如“运行文件”、“测试文件”以及“调试文件”。

24.2 创建自由格式项目

本节将介绍如何使用已经存在的 Ant 脚本，创建自由格式的项目。按照如下步骤完成项目的创建。

- （1）启动 NetBeans，依次选择主菜单的“File”/“New Project”选项，出现如图 24-1 所示的窗口。
- （2）在图 24-1 中的“Projects”列表框中，选中“EJB Module with Existing Ant Script”选项。单击“Next”按钮，如图 24-2 所示。



说明

对于图 24-1 中“Categories”列表框的“General”、“Web”与“Enterprise”选项，在“Projects”列表框中都有相应的“XXX with Existing Ant Script”选项。选中相应的“XXX with Existing Ant Script”选项，即可以使用已有的 Ant 脚本创建相应的项目。在本例中，使用的是已有的 Ant 脚本创建 EJB 模块。

- （3）在如图 24-2 所示的窗口中，可以设置包含项目各元素的目录（通过设置 Location 选项的值完成）。

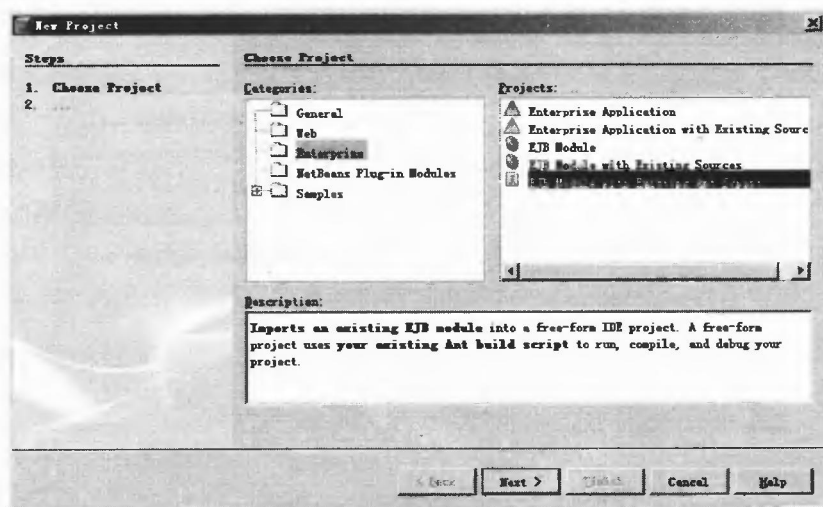


图 24-1 使用已经存在的 Ant 脚本创建 EJB 模块

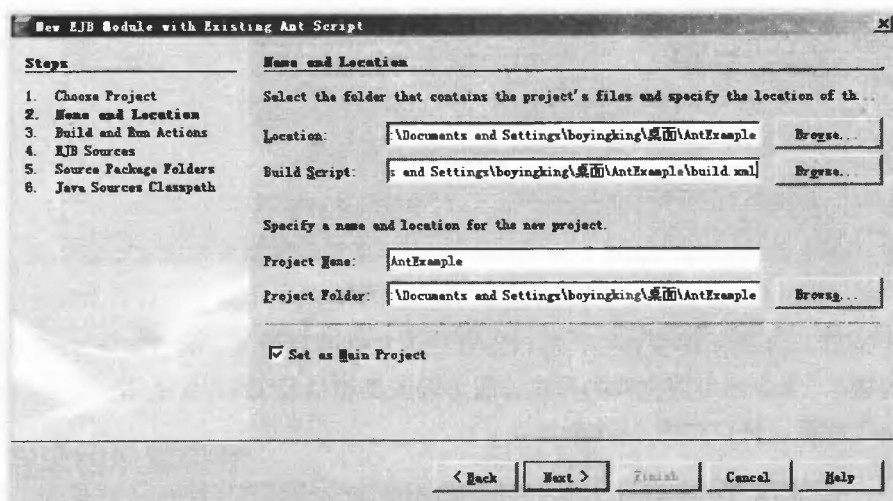


图 24-2 设置项目的名称和位置



说明

如果生成脚本位于指定目录的根目录下，则剩下的字段会自动填入，如果在指定的目录下没有找到生成脚本，则需要手动填写“Build Script”字段。

(4) 在如图 24-2 所示的窗口中，通过设置“Project Name”与“Project Folder”选项的值，来设置将要生成的项目名称及位置。设置完成以后，单击“Next”按钮，窗口如图 24-3 所示。

(5) 在如图 24-3 所示的窗口中，可以单击每个文本框右侧的▼按钮，选择目标；设置完成以后，单击“Next”按钮，窗口如图 24-4 所示。



提示

还可以手动修改相应文本框的值，来指定 NetBeans 命令的目标，这样，当在 NetBeans 中运行命令的时候，可以运行相应的脚本。另外，如果 NetBeans 能够找到近似的目标，将自动填入该目标。

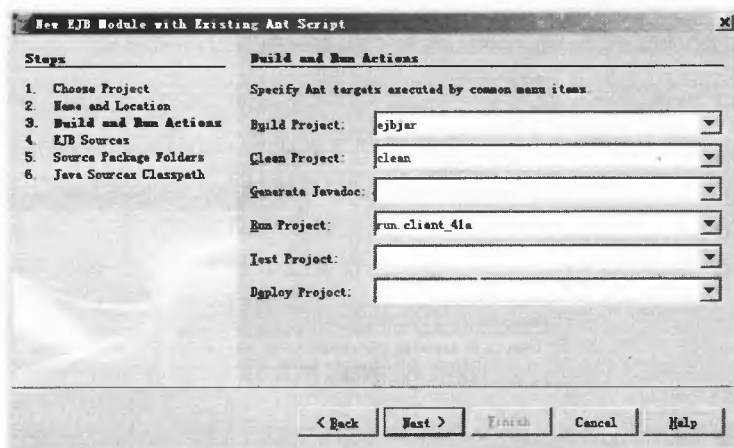


图 24-3 生成与运行窗口

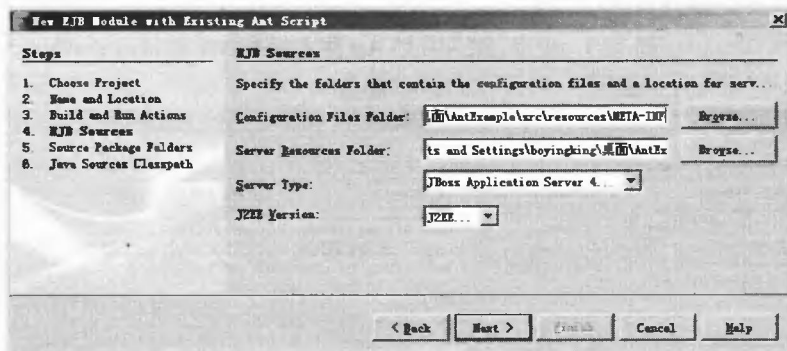


图 24-4 设置 EJB 资源

(6) 在如图 24-4 所示的窗口中，可以设置配置文件的目录（一般情况下，NetBeans 会自动填入该内容）、服务器资源文件的目录、服务器的类型以及 J2EE 的版本。设置完成以后，单击“Next”按钮，窗口如图 24-5 所示。

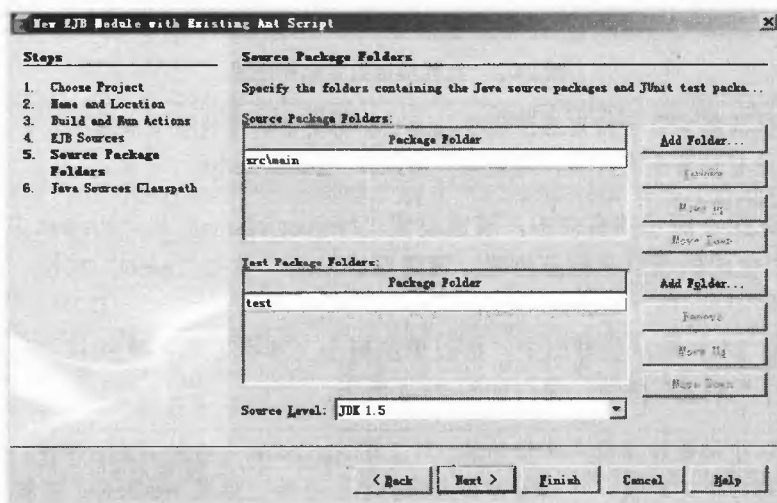


图 24-5 设置资源文件的目录

说明

在如图 24-5 所示的窗口中，可以设置包的顶级目录（一般情况下，NetBeans 会自动设置该内容）。如本例中源代码根的包结构以 main 开始，则选择包含 main 的目录 src。如果存在测试包，也可以通过单击“Test Package Folders”右侧的“Add Folder”按钮进行添加。

(7) 单击“Source Level”右侧的▼按钮，选择 JDK 的版本。设置完成以后，单击“Next”按钮，窗口如图 24-6 所示。

(8) 在如图 24-6 所示的页面中，可以设置编译源文件所需要的库或者其他类文件。设置完成以后，单击“Finish”按钮，完成项目的创建。此时项目窗口中内容如图 24-7 所示。

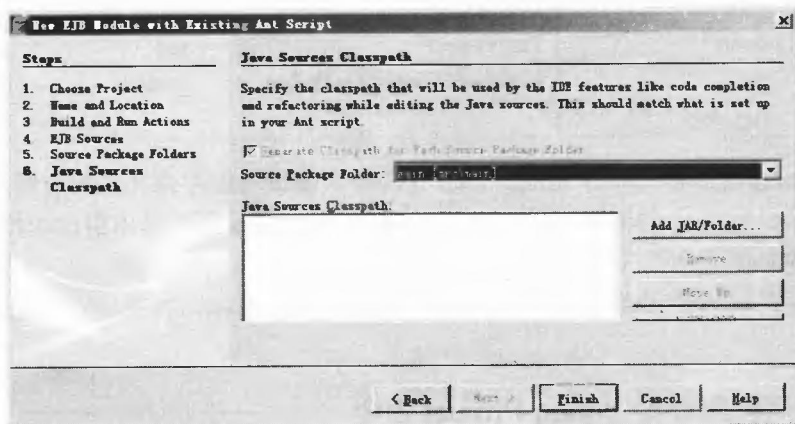


图 24-6 设置 Java 源代码路径

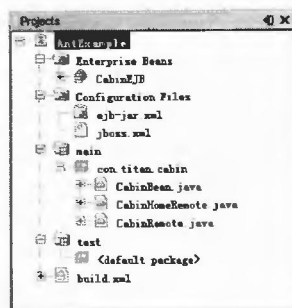


图 24-7 项目窗口的内容

提示

对于在模版向导中设置的大多数字段，都可以在创建项目后，在项目属性对话框中进行设置，因此可以不必在创建项目时设置所有的值。打开项目属性对话框的方法是，用鼠标右键单击项目的主节点（如 AntExample），选择右键菜单的“Properties”选项即可。

24.3 为项目设置命令

本节将为读者介绍创建项目后，如何设置相应的命令，主要包括如下内容：

- 将目标映射到 NetBeans 命令；
- 为 Java 应用程序设置“Debug Project”命令；
- 为 Web 应用程序设置“Debug Project”命令。

24.3.1 将目标映射到 NetBeans 命令

使用已有的 Ant 脚本创建 Java 项目时，在新建项目向导中，可以将特定的目标映射到相应的 NetBeans 命令，这些命令包括：

- Build Project（生成项目）；
- Clean Project（清除项目）；
- Generate Javadoc for Project（生成 Javadoc）；
- Run Project（运行项目）。

对于这些命令，可以在项目属性对话框中，将目标映射到这些命令。而对于其他的命令，如果需要让其正常工作，需要创建对应的目标，并在 `project.xml` 文件中创建映射。表 24-1 列出了可以映射到生成脚本的命令列表，“IDE 动作”列是将生成目标映射到命令时的代码名称。

表 24-1 NetBeans 命令与 `project.xml` 文件中使用的对应动作名称

命 令	IDE 动作	命 令	IDE 动作
Build Project	build	Deploy Project	redeploy
Clean and Build Project	rebuild	Debug Project	debug
Compile File	compile.single	Debug File	debug.single
Clean Project	clean	Test Project	test
Run Project	run	Generate Javadoc for Project	javadoc
Run File	run.single		

NetBeans 绑定了用于 `project.xml` 文件的 XML 方案，并在每次编辑与保存该文件的时候，自动验证该文件的有效性。如果对 `project.xml` 进行无效的调整，NetBeans 会在输出窗口中报告错误。验证有效性的过程会使用若干 XML 方案。

在接下来的几个小节中，将为读者讲述如何为特定命令创建 Ant 目标，并在 NetBeans 中添加命令与目标映射。

24.3.2 为 Java 应用程序设置 Debug Project 命令

为了使自由格式的项目能够正常调试，需要进行如下准备工作。

- 在调用 `Javac` 命令时，确定用于编译的目标中指定了 `debug=true`。
- 在 NetBeans 中，创建项目输出和项目源之间的映射关系，从而保证调试器在进入程序时，可以显示对应的源代码。
- 向 Ant 脚本中添加调试命令的目标，确定该命令所必需的路径元素都在生成脚本中，或者在 `properties` 文件中进行了定义。

准备工作完成以后，即可创建调试目标。生成调试目标时，会产生 `ide-file-targers.xml` 文件，该文件中导入了主生成脚本。如果已经完成了“Run Project”命令的映射，则 NetBeans 生成的调试项目目标如下：

```
<target name="debug-nb">
  <nbgpdastart addressproject="jpda.address" name="ProjectName"
    transport="dt_socket">
    <classpath path="build"/>
  </nbgpdastart>
  <java classname="MainClass" classpath="build" fork="true">
    <jvmarg value="-Xdebug"/>
    <jvmarg value="-Xnoagent"/>
    <jvmarg value="-Djava.compiler=none"/>
    <jvmarg value="-Xrunjdp:transport=dt_socket,address=${jpda.address}"/>
  </java>
</target>
```

在多数情况下，自动生成的目标不需要修改即可正常工作。如果目标不能按照设计的方式工作，可以手动对其进行修改，表 24-2 给出了有关的细节，以及需要注意的事项。

表 24-2 常规 Java 应用程序的调试目标的细节

目标、任务、特性及属性	描 述
netbeans.home	在 NetBeans 内运行的任何 Ant 实例加载的 Ant 属性
nbjpdastart	NetBeans 绑定的特殊任务，使用 JPDA 调试器调试程序
addressproperty	nbjpdastart 的一个属性，定义了指定调试器侦听端口的属性，NetBeans 会自动将分配的端口值赋予该属性
transport	指定需要使用的调试传输协议的属性
classpath	该属性表示用于调试应用程序的类路径
sourcepath	nbjpdastart 的一个可选属性，用来指定对应类路径中 JAR 文件源代码的明确位置
fork	Java 任务的属性，该属性决定调试进程是否在分离的虚拟机中启动，对于调试项目的目标，该属性值为 true
classname	Java 任务的属性，该属性指向调试器执行的类。对于调试项目的目标，该属性应当是项目主类的完全路径。生成调试目标时，NetBeans 会自动填入运行项目目标提供的类名称
jvmarg	Java 元素的参数，为 JVM 提供运行的参数。示例中的参数是典型用于使用 JPDA 调试器调试 J2SE 程序的参数

如果在生成调试目标前，没有在项目指定运行目标，那么在生成的目标中需要手动填入一些内容，这样的目标文件如下：

```
<target name="debug-nb">
  <path id="cp">
    <!--TODO configure the runtime classpath for your project here-->
  </path>
  <nbjpdastart addressproject="jpda.address" name="ProjectName" transport=
"dt_socket">
    <classpath ref id="cp"/>
  </nbjpdastart>
  <!--TODO configure the main class for your project here:-->
  <java classname="some.main.Class" classpath="build" fork="true">
    <classpath ref id="cp"/>
    <jvmarg value="-Xdebug"/>
    <jvmarg value="-Xnoagent"/>
    <jvmarg value="-Djava.compiler=none"/>
    <jvmarg value="-Xrunjdpw:transport=dt_socket,address=${jpda.address}"/>
  </java>
</target>
```

上面的 TODO 注释内容，指定了类路径。对于主类，需要输入其全称类名。可以将 pathelement 元素放置在 path 元素内。例如可以使用 pathelement 的 location 属性来指定相对项目目录（包含 build.xml 文件的目录）的位置，代码如下：

```
<path id="cp">
  <pathelement location="libs">
  <pathelement location="classes">
</path>
```

如果项目非常复杂,可以嵌套 path 元素,其中 pathelement 的 path 属性引用一个不同的 path 元素,代码如下:

```
<path id="run.classpath">
  <pathelement path="${javac.classpath}">
  <pathelement path="${build.classes.dir}">
</path>
```

24.3.3 为 Web 应用程序设置 Debug Project 命令

调试自由格式的 Web 项目时,需要进行以下操作。

- 在调用 javac 命令时,确定用于编译的目标中指定了 debug="true"。
- 在 NetBeans 中,创建项目输出和项目源之间的映射关系,从而保证调试器在进入程序时,可以显示对应的源代码。
- 向 Ant 脚本中添加一个目标,用于将调试器连接到运行的 Web 项目中。确保用于该命令的所有必需的路径元素,在生成的脚本中都已经定义。
- 确保 Web 服务器启动运行在调试模式。
- 确保 Web 应用程序已经部署。

生成调试项目时,会生成一个名称为 ide-file-targets.xml 的文件。该文件导入了主生成脚本,可以帮助开发者将 IDE 专有目标与其他目标区分开。同时还可以使 IDE 特定的目标,能够引用主生成脚本中的目标与属性。ide-file-targets.xml 中主要代码如下:

```
<target name="-load-props">
  <property file="nbproject/debug.properties"/>
</target>
<target name="-check-props">
  <fail unless="jpda.session.name"/>
  <fail unless="jpda.host"/>
  <fail unless="jpda.address"/>
  <fail unless="jpda.transport"/>
  <fail unless="debug.sourcepath"/>
  <fail unless="client.url"/>
</target>
<target depends="-load-props, -check-props" name="-init"/>
<target depends="-init" if="netbeans.home" name="debug-nb">
  <nbjpdacconnect address="${jpda.address}" host="${jpda.host}"
    name="${jpda.session.name}" transport="${jpda.transport}">
    <sourcepath>
      <path path="${debug.sourcepath}"/>
    </sourcepath>
  </nbjpdacconnect>
</target>
```

```
</nbjpdacconnect>
<antcall target="debug-display-browser"/>
</target>
```

另外，调试 Web 应用程序时，还会生成一个名为 `debug.properties` 的文件。该文件的内容如下：

```
jpda.session.name=MyWebProject
jpda.host=localhost
# Sun Java System Application Server using shared memory (on Windows)
# jpda.address=localhost4848
# jpda.transport=dt_shmem
# Sun Java System Application Server using a socket
jpda.address=9009
jpda.transport=dt_socket
# Tomcat using shared memory(on Window)
jpda.address=tomcat_shared_memory_id
jpda.transport=dt_schem
# Tomcat using a socket
jpda.address=11555
jpda.transport=dt_socket
src.folders=src
web.docbase.dir=web
# you can change this property to a list of your source folders
debug.sourcepath=${src.folders}:{web.docbase.dir}
# Client URL for Tomcat
client.url=http://localhost:8877/MyWebProject
```

通过对 `debug.properties` 文件进行适当的修改，自动生成的目标就可以正常工作了。对该目标进行定制后，便可以在项目中使用。表 24-3 给出了 `debug.properties` 的详细解释。

表 24-3 `debug.properties` 的详细解释

调 试 属 性	说 明
jpda.session.name	调试应用程序的时候，在会话窗口中显示的名称
jpda.host	运行调试中应用程序的机器名称
jpda.address	调试器所侦听的端口号码
jpda.transport	JPDA 调试所使用的传输协议。可以在所有平台上使用 <code>dt_socket</code> 。在 Windows 平台下，还可以使用 <code>dt_schem</code>
src.folders	Java 源文件的位置
web.docbase.dir	Web 应用程序的根目录
debug.sourcepath	调试过程中所引用的源代码的位置。默认情况下，该值是对 <code>src.folders</code> 与 <code>web.docbase.dir</code> 属性的引用
client.url	NetBeans 指定的默认浏览器中打开的网页

表 24-4 给出了调试目标的详细解释。

表 24-4

调试目标的详细解释

目标、任务、特性及属性	说 明
depends	用于指定当前目标运行前所运行的目标的属性
netbeans.home	运行于 NetBeans 中的任何 Ant 实例加载的 Ant 属性。If=“netbeans.home”属性保证了目标只在 NetBeans 内部调用时才运行
nbpdaconnect	NetBeans 绑定的特殊任务，用于 JPDA 调试器连接到应用程序
host	nbpdaconnect 的属性，用来指定运行调试程序的机器名称
address	nbpdaconnect 的属性，用来指定调试器所侦听的端口
transport	用于指定 JPDA 调试所使用的传输协议。可以在所有平台上使用 dt_socket。在 Windows 平台下，还可以使用 dt_schem
classpath	nbpdaconnect 可选的属性，表示调试应用程序所使用的类路径
sourcepath	nbpdaconnect 的属性，用于指定对应类路径中 JAR 文件的源文件位置
nbbrowse	指定浏览器打开的 Web 页面的元素

24.4 为文件设置命令

上一节中介绍了如何为项目设置命令，本节中将介绍如何为文件设置命令。主要包括如下内容：

- 为文件设置命令的说明；
- 设置“Compile File”命令；
- 设置“Run File”命令；
- 设置“Debug File”命令。

24.4.1 为文件设置命令的说明

为了使文件相关的命令（如编译文件、运行文件以及调试文件）在 NetBeans 中正常工作，需要进行以下操作。

（1）向 Ant 脚本中添加命令的对应目标，确保该命令必需的路径元素在生成脚本，或者生成脚本所调用的.properties 文件中进行定义。

（2）在项目的 project.xml 文件中，将目标映射到 NetBeans 命令，并且添加 context 元素，提供将当前选中的文件传送到 Ant 脚本的方式。如果目标是 NetBeans 自动生成的，那么这些操作也会自动完成。

（3）在 project.xml 文件中，定义 project.xml 文件需要的所有属性。

在接下来的几个小节中，将为读者详细地讲述如何进行相应命令的设置。

24.4.2 设置 Compile File 命令

在 NetBeans 的自由格式项目中，编译选中的文件，会创建对应的“Compile File”命令的目标。此时，NetBeans 在 ide-file-targets.xml 文件中，会生成一个目标框架，并且在 project.xml 文件中创建目标命令与 IDE 命令之间的映射。目标框架的代码如下：

```

<target name="compile-selected-files-in-src">
    <fail unless="files"/>Must set property 'file'</fail>
    <mkdir dir="build"/>
    <javac destdir="build" includes="${files}" source="1.5" srcdir="src">
        <classpath path="resource"/>
    </javac>
</target>

```

在以上代码中，files 属性获取 NetBeans 中选中的文件。选择“Compile File”命令的时候，files 属性值从 project.xml 文件中获取。如果使用“Compile File”命令时没有选中任何文件，即没有任何值传送到 files 属性，目标就无法成功创建。

在生成脚本中的目标，通过 project.xml 文件中的<action>元素，会映射到 NetBeans 命令。当 NetBeans 自动生成命令对应的目标时。映射会自动添加到 project.xml 文件中。如果手动编写目标。则同样需要手动添加映射。

表 24-5 列出了 project.xml 文件中<action>元素各部分的说明。

表 24-5 project.xml 文件中<action>元素的说明

目标、任务或者属性	说 明
context	用于收集命令运行的目标文件信息的参数
property	该参数定义了执行当前命令时，用于传递当前选中文件名的属性名称，目标可以引用该属性来确定命令执行的目标文件
folder	用于指定启用了 filesselected-compile 目标的目录，在上面 compile-selected-files-in-src 示例中，该值被用作到 src 的属性
pattern	该参数包含了一个正则表达式，用于限制可以运行目标的文件类型。例如，只有具有 .java 扩展名的文件被传送到目标
format	该参数指定了选中文件被传送到目标的格式。该元素的可能值有： relative-path—传送相对于 folder 元素的相对路径文件名 relative-path-noext—与 relative-path 相似，不过文件不带扩展名 java-name—与 relative-path-noext 类似，不过使用斜线“/”而不是“.”来分隔路径中的目录 absolute-path—传递使用绝对路径的文件名 absolute-path-noext—与 absolute-path 类似，不过没有扩展名
arity	该参数用于确定单个文件还是多个文件可以被传送到目标

在本节给出的代码示例中，src 属性在 project.xml 文件中被引用。该属性需要在 project.xml 文件中，或者该文件引用的文件中定义。

在 project.xml 文件中，属性定义在<properties>元素中，该元素属于<name>与<folder>之间的元素。在<properties>元素中，使用<properties>元素与 name 属性来定义单个属性，或者使用<property-file>元素来指定 properties 文件。

完成新建项目向导，指定使用的生成脚本后，project.xml 文件中会生成如下内容：

```

<properties>
    <property name="project.dir">C:\SampleProjects\MyProject</property>
    <property name="ant.script">${project.dir}/build.xml</property>
</properties>

```

在<properties>元素中，可以添加更多的属性或者属性文件，因为在本示例中的 src 属性可能也要在生成脚本中使用，所以如果在某个位置设置该属性，让生成脚本与 project.xml 文件都使用它会带来很多方便。

project.xml 文件中，对某个.properties 文件的引用如下：

```
<property-file>${project.dir}/MyProject.properties</property-file>
```

project.xml 文件中，引用的文件路径都是相对于项目目录的路径。如果需要使用
提示 project.xml 文件与生成脚本都能使用路径引用，生成脚本必须存放在项目目录下（实际上是包含 nbproject 目录的目录）。

24.4.3 设置 Run File 命令

前面讲述了如何设置“Compile File”命令，本节将介绍如何设置“Run File”命令。在 NetBeans 的自由格式项目中运行选定的文件，需要创建对应运行文件命令的目标，并在 project.xml 文件中创建目标到命令的映射。

1. 创建 run-selected-file 目标

以下是运行选中文件的示例目标：

```
<target name="run-selected-file">
  <depends="compile-selected-files-in-src"
  description="Run Single File"/>
  <fail unless="selected-file">
    Must Set Property 'selected-file'
  </fail>
  <java classname="${selected-file}">
    <classpath ref id="run.classpath"/>
  </java>
</target>
```

在本示例中，selected-file 属性获取 NetBeans 中选中的文件。选择“Run File”命令的时候，selected-file 属性的值从 project.xml 文件中被传送出。本例还假设已经存在 compile-selected-files-in-src 目标，尽管可以将本目标依赖于脚本中设定的不同编译目标。

在本例中，使用 refid 属性来引用脚本，在其他位置定义的运行类路径，其中指定了 run.classpath 作为 path 元素 ID 属性的值。run.classpath 可以按照下文所示定义：

```
<path id="run.classpath">
  <pathelement path="${javac.classpath}">
  <pathelement path="${build.classes.dir}">
</path>
```

javac.classpath 与 build.classes.dir 的路径需要自己的路径元素中定义。如果路径元素需要引用一个物理位置，则 location 元素可以用于指定一个相对于 Ant 项目目录的路径。代码如下：

```
<path id="build.classes.dir">
  <pathelement location="classes">
</path>
```

2. 将 run-selected-file 目录映射到 IDE 命令

将 run-selected-file 目录映射到 IDE 命令非常简单，只需打开 project.xml 文件，在 <ide-action>元素中，添加目标到“Run File”命令的映射即可。该映射代码如下：

```
<action name="run.single">
  <target>run-selected-file</target>
  <context>
    <property>selected-file</property>
    <folder>${src.dir}</folder>
    <pattern>\.java$</pattern>
    <format>java-name</format>
    <arity><one-file-only/></arity>
  </context>
</action>
```

提示 对于每个元素的详细说明，可以参照表 24-5。

24.4.4 设置 Debug File 命令

在 NetBeans 自由格式项目中调试选中的文件，需要创建对应调试文件的目标，并在 project.xml 文件中创建目标到命令的映射。

1. 创建 debug-selected-file 目标

以下是调试选中文件的目标示例代码：

```
<target name="debug-selected-file"
  depends="compile-selected-files-in-src" if="netbeans.home"
  description="Debug a Single File">
  <fail unless="selected-file">
    Must Set Property 'selected-file'
  </fail>
  <nbpdaconnect name="${selected-file}" addressproperty="jpda.address"
    transport="dt_socket">
    <classpath refid="run.classpath"/>
    <sourcepath refid="debug.sourcepath"/>
  </nbpdaconnect>
  <java classname="${selected-file}" fork="true">
    <classpath refid="run.classpath"/>
    <jvmarg value="-Xdebug"/>
    <jvmarg value="-Xnoagent"/>
    <jvmarg value="-Djava.compiler=none"/>
    <jvmarg value="-Xrunjdp:transport=dt_socket,address=${jpda.address}"/>
  </java>
</target>
```

在本示例中，selected-file 属性获取 NetBeans 中选中的文件。选择“Debug File”命令的时候，selected-file 属性的值从 project.xml 文件中传出。

本示例使用了 refid 属性来引入两个路径元素（run.classpath 与 debug.sourcepath），这两个元素需要在生成脚本的其他位置定义。例如，run.classpath 元素可以由以下代码定义：

```
<path id="run.classpath">
  <pathelement path="${javac.classpath}">
  <pathelement path="${build.classes.dir}">
</path>
```

javac.classpath 与 build.classes.dir 的路径，需要在其自己的路径元素中定义。如果路径元素需要引用一个物理位置，则 location 元素可以用于指定一个相对于 Ant 项目目录的路径。代码如下：

```
<path id="build.classes.dir">
  <pathelement location="classes">
</path>
```

2. 将 debug-selected-file 目标映射到 IDE 命令

将 debug-selected-file 目录映射到 IDE 命令非常简单，只需打开 project.xml 文件，在 <ide-action>元素中，添加目标到“Debug File”命令的映射即可。该映射代码如下：

```
<action name="debug.single">
  <target> debug -selected-file</target>
  <context>
    <property>selected-file</property>
    <folder>${src.dir}</folder>
    <pattern>\.java$</pattern>
    <format>java-name</format>
    <arity><one-file-only/></arity>
  </context>
</action>
```

提示 对于每个元素的详细说明，可以参照表 24-5。

24.5 更改自由格式项目的目标 JDK

前面为读者介绍了如何设置编译、运行、调试文件的命令，本节将介绍如何设置自由格式项目的 JDK。如果设置与 NetBeans 所运行的 JDK 不同的目标 JDK，必须在以下两个位置都指定 JDK 版本。

- 任何与项目命令相关任务的 Ant 脚本，如 javac。它将确保生成的目标（以及调用这些目标的 IDE 命令）能够正确运行。
- 项目的属性对话框。它将确保 NetBeans 的功能，比如代码自动完成功能弹出的 Javadoc 窗口功能能够正确的工作。

对于 Javac 任务,可以在调用 javac 命令的时候,使用 source 与 target 选项来更改目标 JDK。例如 javac 任务的调用可以按照下面的示例进行。对于 javac.source 与 javac.target 属性,需要在生成脚本或者某个 .properties 文件中指定,指定的值应该是适当的 JDK 版本(如 1.3、1.4 或者 1.5)。

```
<javac srcdir="${src.dir}" destdir="${classes.dir}"  
    debug="true" source="${javac.source}"  
    target="${javac.target}">  
    <classpath refid="javac.classpath"/>  
</javac>
```

在项目窗口中更改目标 JDK 的步骤如下。

(1) 用鼠标右键单击要更改目标 JDK 的项目,在弹出的菜单中选择“Properties”选项,接着在弹出窗口的“Categories”列表框中选中“Java Sources”选项,如图 24-8 所示。

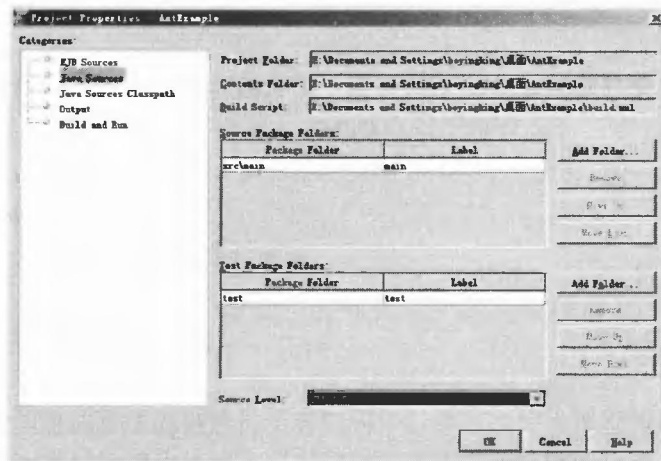


图 24-8 更改目标 JDK

(2) 单击“Source Level”右侧的▼按钮,选择使用的目标 JDK (JDK1.3、JDK1.4、JDK1.5)。选中相应的目标 JDK 按钮后,单击“OK”按钮即可。

24.6 小结

本章主要为读者讲述了如何使用 NetBeans 集成已存在的 Ant 脚本。包括如何创建自由格式的项目、如何为项目设置命令、如何为文件设置命令,以及如何更改自由格式项目的目标 JDK。通过集成已经存在的 Ant 脚本,可以更多地使用在其他集成开发环境中创建的项目,增强了项目的可重用性。

第 25 章

在 NetBeans 中使用 JUnit

测试是开发过程中一个非常重要的环节，良好的测试是保证软件质量的基石之一，在本章中将对 Java 测试方面的知识进行介绍，主要包括以下内容：

- 什么是单元测试；
- JUnit 介绍；
- 在 NetBean 中使用 JUnit 进行测试。

25.1 什么是单元测试

单元测试是指将测试集成到创建的所有代码中，并且在每次执行构建时运行这些测试的过程。这样，在每次进行构建的过程中，不仅可以进行语法错误的检查，而且可以方便地进行语义错误的检查。这样可以很好地保证开发代码的正确性，从而方便了开发。

在 Java 中，如果需要进行单元测试，开发人员可以开发自己的单元测试框架。但是开发一个好的单元测试框架，是非常耗费时间的。现在已经有了很多非常优秀的用于 Java 的单元测试框架，开发人员可以根据自己的需要选用。常用的单元测试框架有：

- Kent Beck 和 Erich Gamma 的 Junit；
- Bill Venners 的 Artima SuiteRunner；
- Cedric Beust 的 TestNG。

Kent Beck 和 ErichGamma 开发的 JUnit，在单元测试框架中的占有率较高，在本章的后续部分将对 JUnit 进行详细的介绍。

25.2 JUnit 简介

JUnit 是一个开源的 Java 单元测试框架。在 1997 年，由 Erich Gamma 和 Kent Beck 开发完成。JUnit 设计的非常小巧，但是功能却非常强大。下面是 JUnit 一些特性的总结：

- 提供了 3 种方式来显示测试结果，而且还可以扩展；

- 提供了单元测试用例成批运行的功能；
- 超轻量级而且使用简单；
- 整个框架设计良好、易扩展。

读者需要注意的是，对不同性质的被测对象，如 Class、JSP、Servlet、EJB 等，JUnit 有不同的使用技巧。JUnit 是免费的，可以去官方网站 <http://www.junit.org> 下载最新版本，本书将以 3.8.1 版为例进行介绍。

25.3 TestCase 类简介

在使用 JUnit 开发自己的测试用例（TestCase）时，需要继承由 JUnit 系统提供的 `junit.framework.TestCase` 类。TestCase 有几个重要的方法需要进行重写，如表 25-1 所示。

表 25-1 TestCase 类中需要重写的方法

方 法	说 明
<code>runTest()</code>	重写此方法来运行测试用例
<code>setUp()</code>	在此方法中写入对测试用例进行准备的工作，如连接数据库、日志文件等，在每次运行测试前都会自动调用该方法
<code>tearDown()</code>	在此方法中写入测试结束后释放在 <code>setUp</code> 中创建资源的代码，如释放网络连接、数据库连接等，在每次运行测试后都会自动调用该方法

TestCase 类还提供了一系列用来进行测试的静态方法 `assertXXX`，在编写具体的测试时，开发人员可以使用这些方法来检查要测试的条件是否满足。如果测试没有通过，这些方法会抛出 `AssertionFailedError` 异常，向 JUnit 系统报告测试没有通过的原因。

在表 25-2 中列出了其中常用的一些方法。

表 25-2 TestCase 类中提供的静态测试方法

方 法	说 明
<code>assertEquals(od1,od2)</code>	这个方法被重载了，具有很多版本，任何两个类型的变量 <code>od1</code> 和 <code>od2</code> 都可以进行比较，要求 <code>od1</code> 和 <code>od2</code> 在 Java 中具有可比性即可。如果比较结果为 <code>false</code> ，就会向执行测试的 JUnit 系统报告
<code>assertEquals(java.lang.String message,od1,od2)</code>	这个方法被重载了，具有很多版本，任何两个类型的变量 <code>od1</code> 和 <code>od2</code> 都可以进行比较，要求 <code>od1</code> 和 <code>od2</code> 在 Java 中具有可比性即可。如果比较结果为 <code>false</code> ，就会向执行测试的 JUnit 系统报告，报告时采用由参数 <code>message</code> 提供的信息
<code>assertFalse(boolean condition)</code>	测试传入的参数是否为 <code>false</code>
<code>assertNull(Object object)</code>	测试传入的句柄是否为 <code>null</code>
<code>assertSame(Object expected,Object actual)</code>	测试传入的两个句柄是否指向同一个对象

下面通过一个简单实例来说明如何使用 TestCase 类开发一个对某个类的测试。首先开发一个用于被测试的 `StudentBean` 类，下面给出了这个类的源代码（`StudentBean.java`）：

```
public class StudentBean {  
    private String sname;  
    private int sage;
```

```
public void setName(String sname){
    this.sname=sname;
}
public String getName(){
    return sname;
}
public void setSage(int sage){
    this.sage=sage;
}
public int getSage(){
    return sage;
}
}
```

这是一个非常简单的 JavaBean 代码，具有 `sname` 和 `sage` 两个私有属性，并且每个属性都有对应的 `set` 和 `get` 方法。下面就开发一个自己的测试用例 `MyTestCase1`，用来对 `StudentBean` 进行测试，程序代码如下（`MyTestCase1.java`）：

```
import junit.framework.*;
public class MyTestCase1 extends TestCase{
    public void testSname(){
        System.out.println("testSname");
        String sname="Tom";
        StudentBean sb=new StudentBean();
        sb.setName(sname);
        String result=sb.getName();
        /*测试 age 和 result 是否相等，如果相等说明
        *setSage 和 getSage 方法工作正常
        */
        assertEquals(sname, result);
    }
    public void testSage(){
        System.out.println("testSage");
        int sage=23;
        StudentBean sb=new StudentBean();
        sb.setSage(sage);
        int result=sb.getSage();
        /*测试 age 和 result 是否相等，如果相等说明
        * setName 和 getName 方法工作正常
        */
        assertEquals(sage, result);
    }
    //在此方法中写上每次测试前的准备代码
    public void setUp(){
        System.out.println("SetUP");
    }
    //在此方法中写上每次测试后的释放资源代码
```

```
public void tearDown(){
    System.out.println("TearDown");
}
}
```

- 每一个测试应该写成一个名称以“test”开头的方法，并且必须是 public 的，返回值为 void。
- testName 方法用来测试 StudentBean 中 setName 和 getName 方法的正确性。
- testSage 方法用来测试 StudentBean 中 setSage 和 getSage 方法的正确性。

在编写完测试用例的代码后，对其进行编译，如果没有错误，就可以使用 JUnit 来运行测试了。可以使用如下命令来运行测试：

```
java -classpath H:\ygq\java_c\junit\junit-3.8.1.jar;H:\ygq\java_c\l_4
java\JUnit_test junit.swingui.TestRunner MyTestCase1
```

- H:\ygq\java_c\junit\junit-3.8.1.jar 为下载的 JUnit jar 文件的路径。
- H:\ygq\java_c\l_4java\JUnit_test 为测试用例类所在的路径。
- MyTestCase1 为要运行的测试用例的类名。

提示 读者可以把该命令写到一个 run.bat 或 run.sh 文件中，这样在运行测试时就方便了。该例的所有代码在本书赠送光盘 26 章的 JUnit_test 目录中。

在键入了以上命令后，系统会弹出测试运行的结果窗口，如图 25-1 所示。

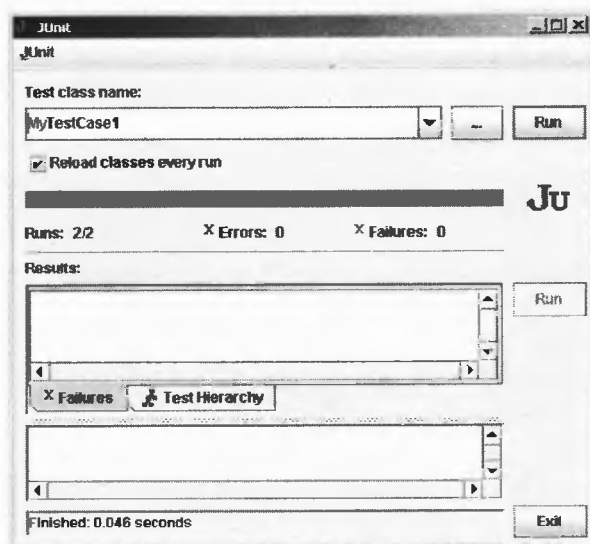


图 25-1 JUnit 运行窗口

- Runs: 2/2 表示一共运行了两个测试。JUnit 会自动运行指定的测试用例类的所有测试，在本例中正好是两个。
- X Errors: 0 表示测试运行中产生的错误。“Errors”不是期待的错误，是指在运行测试的过程中本身产生了错误，而不是某个测试没有通过。
- X Failures: 0 表示成功运行的测试中没有通过测试的数量。“Failures”是作为单元测试

所期望发生的错误，预示代码有 bug。

- 如果有没有通过的测试项目，在窗口的“Result”及下面的文本框中会显示详细的原因。

细心的读者还会发现，在键入了命令成功运行了测试以后，控制台窗口中还会打印两次“SetUp”和“TearDown”，如图 25-2 所示。这说明每执行一次测试 JUnit 系统都会在测试前调用 setUp()方法，在测试后调 tearDown()方法。

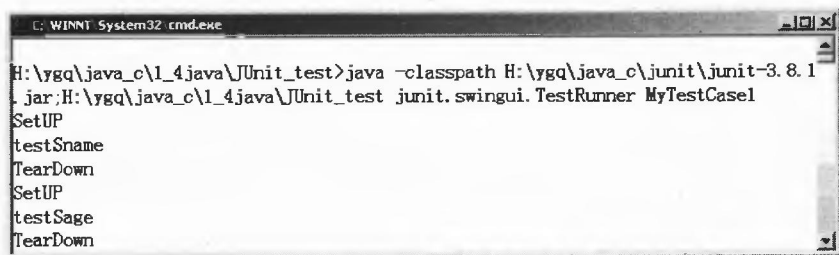


图 25-2 执行测试后的控制台窗口

25.4 TestSuite 类简介

在实际测试工作中，可能需要一次执行多个测试示例，如果每次测试都需要逐个进行，就比较烦琐。JUnit 向开发人员提供了一个名称为 TestSuite 的类，其属于 junit.framework 包，可以用来开发测试套件。利用它就可以依次运行多个测试示例了。

TestSuite 类有很多不同的构造器，用来在不同情况下创建 TestSuite 对象，表 25-3 列出了其中常用的几个构造器。

表 25-3 TestSuite 类的常用构造器

方 法	说 明
TestSuite()	创建一个空的测试套件对象
TestSuite(java.lang.Class theClass)	创建对应指定测试用例类的测试套件对象
TestSuite(java.lang.Class theClass,java.lang.String name)	创建一个对应指定测试用例类和给定名称的测试套件对象

TestSuite 类还有很多方法可以对其进行操作，表 25-4 列出了其中的一部分。

表 25-4 TestSuite 类的常用方法

方 法	说 明
addTestSuite(java.lang.Class testClass)	向测试套件中加入其他套件
addTest(Test test)	向测试套件中加入测试用例
getName()	获取测试套件的名称
setName(java.lang.String name)	设置测试套件的名称
testCount()	返回测试套件中测试的个数

读者如果需要用到其他功能可以查阅 JUnit 相关的 API，由于篇幅所限，这里就不对所有的方 法一一罗列了。下面通过一个例子说明如何开发一个测试套件，下面列出了一个使用

测试套件的程序代码 (MyTestAll.java):

```
import junit.framework.*;
public class MyTestAll {
    public static Test suite() {
        TestSuite myTestSuite=new TestSuite("WYF Test Suite");
        myTestSuite.addTestSuite(MyTestCasel.class);
        myTestSuite.addTestSuite(MyTestCasel.class);
        return myTestSuite;
    }
}
```

- 自己开发的测试套件类一定要有一个 public static Test suite()的方法。
- 在用 JUnit 运行此类时, JUnit 会自动寻找该方法, 获得测试套件 (TestSuite) 对象, 并执行里面的所有测试。
- 在该方法中写入生成需要的测试套件的代码。

在程序编写完成后对其进行编译, 如果编译无误, 用以下命令执行此测试套件:

```
java -classpath H:\ygq\java_c\junit\junit-3.8.1.jar;H:\ygq\java_c\1_4
java\JUnit_test junit.swingui.TestRunner MyTestAll
```

- H:\ygq\java_c\junit\junit-3.8.1.jar 为下载的 JUnit jar 文件的路径。
- H:\ygq\java_c\1_4java\JUnit_test 为测试套件类所在的路径。
- MyTestAll 为要运行的测试套件所在的类名。

在键入了以上命令后, 系统会弹出 JUnit 运行指定测试套件后的结果窗口, 如图 25-3 所示。从图中可以看出, 由于向测试套件中添加了两个测试示例, 每个测试示例中有两个测试, 所以一共运行了 4 个测试。从这个例子中可以看到, 用测试套件一次运行多个测试是非常方便的。

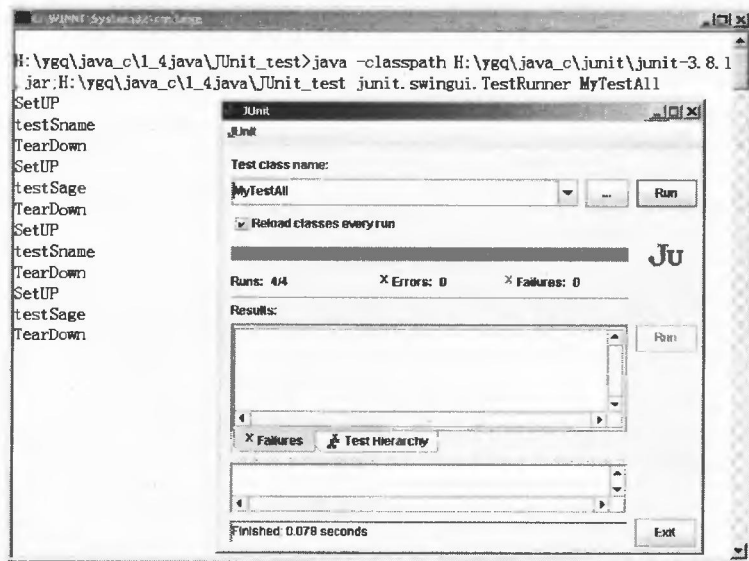


图 25-3 运行测试套件后的 JUnit 窗口和控制台窗口

25.5 在 NetBeans 中使用 JUnit 进行测试

在 NetBeans 5.0 中已经集成了 Junit 3.8.1, 所以如果安装了 NetBeans, 就不需要重新下载安装 JUnit 了。同时, 在 NetBeans 中也集成了很多使用 JUnit 进行测试的功能, 在本小节中将通过一个具体的示例进行介绍, 可按照如下步骤进行操作。

- 创建测试类;
- 查看自动生成的代码;
- 修改测试代码;
- 在 NetBeans 中开发测试套件。

25.5.1 创建测试类

本节将开发 JUnit 测试用到的类, 按照如下步骤完成该类的创建。

(1) 在 NetBeans 中创建一个名称为 NetBeansJUnit 的项目, 项目的类型选择“General”/“Java Application”, 在创建项目时把“Create Main Class”选项去掉。

(2) 在创建完项目后, 在项目窗体中依次展开“NetBeansJUnit”/“Test Libraries”节点, 可以看到“Test Libraries”节点下有一个“JUnit-junit-3.8.1.jar”节点, 如图 25-4 所示。

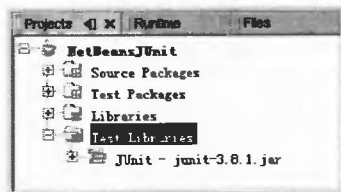


图 25-4 JUnit 节点

提示 NetBeans 中创建的项目默认都已经带有 JUnit 的测试包 junit-3.8.1.jar, 如果不需要可以在该节点上单击鼠标右键, 选择右键菜单中的“Remove”选项将其删除。

(3) 在项目窗口中, 依次单击“NetBeansJUnit”/“Source Packages”节点, 用鼠标右键单击该节点, 在弹出的菜单中依次选择“New”/“Java Class...”选项, 向项目中添加一个名称为“BookBean”的 Java 类。

(4) 完成以后在此类中添加两个成员变量, 具体属性如表 25-5 所示。

表 25-5 两个成员变量的信息

名 称	访 问 限 制	类 型
bookISBN	private	java.lang.String
bookName	private	java.lang.String

(5) 在代码编辑其中右键单击 bookISBN 成员变量, 在弹出的菜单中依次选择“Refactor”/“Encapsulate Field...”选项, 系统将弹出封装字段向导窗口, 如图 25-5 所示。

(6) 在向导窗口中, 选中对应于“bookName”的两个复选框, 表示也要对 bookName 属性进行封装。

(7) 单击“Next”按钮, 在 NetBeans 的窗口中会出现代码重构界面, 如图 25-6 所示。

(8) 在代码重构界面中列出了将要进行的重构动作, 在确认无误后, 单击“Do Refactoring”按钮执行重构。

(9) 在执行完重构动作后, 代码编辑器的源代码中, 出现了两个属性对应的 set 和 get 方法, 如图 25-7 所示。

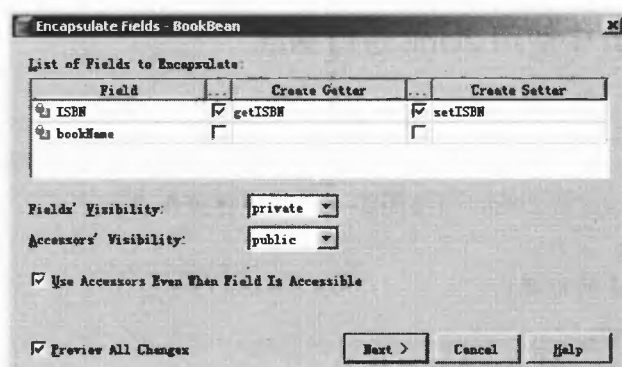


图 25-5 封装字段向导窗口

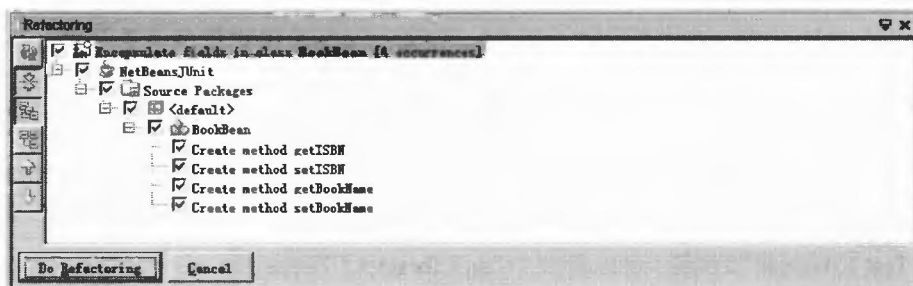


图 25-6 代码重构界面

```

5 public String getISBN() {
6     return ISBN;
7 }
8
9 public void setISBN(String ISBN) {
10     this.ISBN = ISBN;
11 }
12
13 public String getBookName() {
14     return bookName;
15 }
16
17 public void setBookName(String bookName) {
18     this.bookName = bookName;
19 }

```

图 25-7 重构后的代码编辑器



提示

用 NetBeans 的重构 (Refactor) 功能对指定的属性进行封装是非常方便的, 在适当的时候使用, 可以大大提高开发的速度。

(10) 依次选择主菜单中的“File”/“New File...”选项, 在弹出的新建文件向导窗口中设置文件的“Categories”为“JUnit”, “File Types”为“Test for Existing Class”, 如图 25-8 所示。



提示

Test for Existing Class 意味着将要创建的测试类是对已经存在于项目中类的测试, 这样系统会通过向导帮助开发人员自动生成一些代码, 提高开发效率。

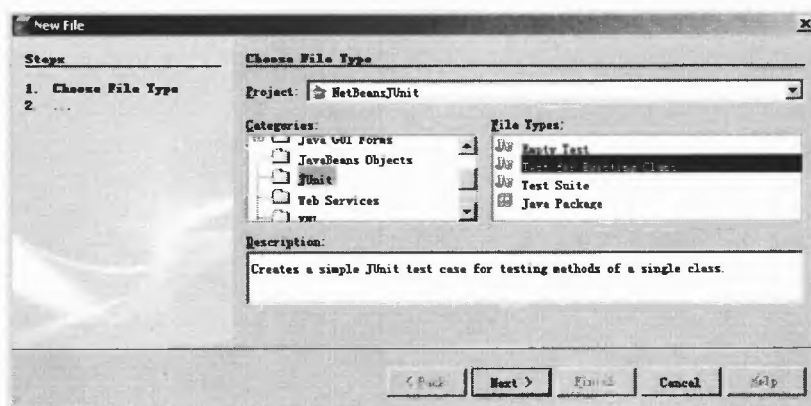


图 25-8 新建文件向导窗口

(11) 单击“Next”按钮，进入选择测试类的向导窗口，如图 25-9 所示。

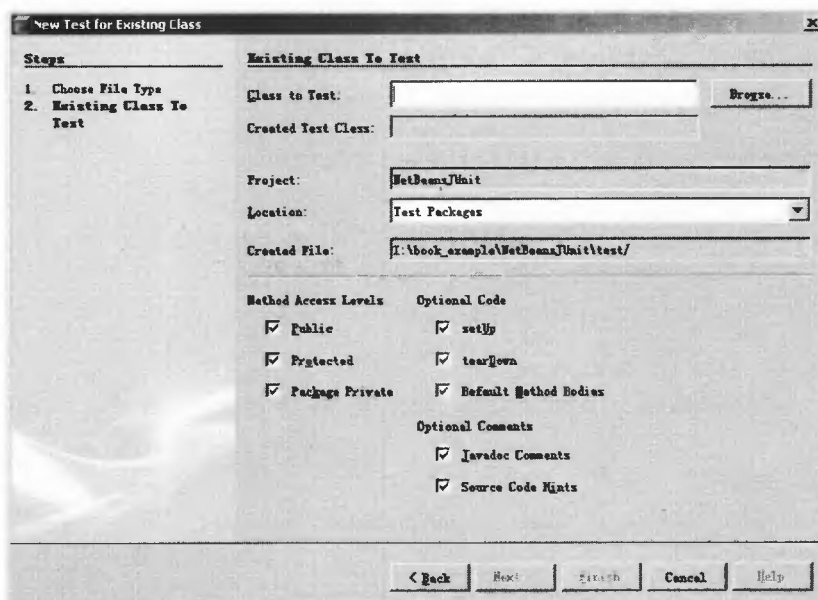


图 25-9 选择测试类向导窗口

(12) 在此窗口中单击“Browse”按钮，系统会弹出选择测试类的列表窗口，如图 25-10 所示。

(13) 在此窗口中选择要被测试的类 BookBean，单击“OK”按钮返回选择测试类向导窗口，然后再单击“Finish”按钮完成。

完成了上述步骤，就已经完成了测试类的创建。

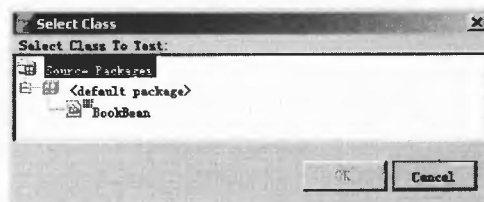


图 25-10 选择测试类的列表窗口

25.5.2 查看自动生成的代码

在上一个小节中，创建了测试需要用到的类。此时，NetBeans 已经自动生成了大部分测

试代码，此时代码编辑器中代码如下：

```
import junit.framework.*;
/*
 * BookBeanTest.java
 * JUnit based test
 *
 * Created on 2006年4月5日, 下午9:30
 */

/**
 *
 * @author Administrator
 */
public class BookBeanTest extends TestCase {

    public BookBeanTest(String testName) {
        super(testName);
    }

    protected void setUp() throws Exception {
    }

    protected void tearDown() throws Exception {
    }

    public static Test suite() {
        TestSuite suite = new TestSuite(BookBeanTest.class);

        return suite;
    }

    /**
     * Test of getISBN method, of class BookBean.
     */
    public void testGetISBN() {
        System.out.println("getISBN");

        BookBean instance = new BookBean();

        String expResult = "";
        String result = instance.getISBN();
        assertEquals(expResult, result);

        // TODO review the generated test code and remove the default
        // call to fail.
        fail("The test case is a prototype.");
    }
}
```

```
}

/**
 * Test of setISBN method, of class BookBean.
 */
public void testSetISBN() {
    System.out.println("setISBN");

    String ISBN = "";
    BookBean instance = new BookBean();

    instance.setISBN(ISBN);

    // TODO review the generated test code and remove the default
    // call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of getBookName method, of class BookBean.
 */
public void testGetBookName() {
    System.out.println("getBookName");

    BookBean instance = new BookBean();

    String expResult = "";
    String result = instance.getBookName();
    assertEquals(expResult, result);

    // TODO review the generated test code and remove the default
    // call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of setBookName method, of class BookBean.
 */
public void testSetBookName() {
    System.out.println("setBookName");

    String bookName = "";
    BookBean instance = new BookBean();

    instance.setBookName(bookName);

    // TODO review the generated test code and remove the default
```

```

        // call to fail.
        fail("The test case is a prototype.");
    }
}

```

- 对于要被测试的 BookBean 类中的 4 个方法, NetBeans 都自动编写了测试代码框架。
- 开发人员可以根据需要对自动生成的测试代码进行适当的修改。

25.5.3 修改并运行测试程序

在上一个小节中, NetBeans 已经自动生成了用于测试的代码。但是其只给出了代码框架, 如果要进行测试, 还需要修改 BookBeanTest 类的代码, 修改后的代码如下:

```

import junit.framework.*;
public class BookBeanTest extends TestCase {
    BookBean book=null;
    public BookBeanTest(String testName) {
        super(testName);
    }
    //调用超类的 setUp(), 以确保测试环境被初始化
    protected void setUp() throws Exception {
        super.setUp();
        System.out.println("测试开始! ");
        book=new BookBean();
        System.out.println("BookBean 对象被初始化! ");
    }
    protected void tearDown() throws Exception {
        System.out.println("BookBean 对象将被清理! ");
        book = null;
        System.out.println("测试结束! ");
        //调用超类的 tearDown(), 以确保测试环境被清理
        super.tearDown();
    }
    public static Test suite() {
        TestSuite suite = new TestSuite(BookBeanTest.class);
        return suite;
    }
    public void testISBN() {
        book.setISBN("ISBN 7-5053-8391-4");//设置 ISBN 属性的值为 ISBN
        //7-5053-8391-4
        //使用 Assert 查看 id 属性的值是否为 ISBN 7-5053-8391-4
        Assert.assertEquals("ISBN 7-5053-8391-4", book.getISBN());
        System.out.println("ISBN 属性被测试! ");
    }
    public void testBookName() {
        book.setBookName("Java 数据库编程宝典");//设置 ISBN 属性的值为"Java
        //数据库编程宝典"
    }
}

```



```
//使用 Assert 查看 id 属性的值是否为"Java 数据库编程宝典"
Assert.assertEquals("笑嘻嘻", book.getBookName());
System.out.println("BookName 属性被测试!");
    }
}
```

- testISBN 方法和 testBookName 方法, 分别用来测试对 ISBN 属性以及 testBookName 属性的访问方法的正确性。
- testISBN 测试会通过, 因为测试的值和设置的值相同。
- testBookName 测试不会通过, 因为测试的值和设置的值不同。
- 这个用例执行后会出现一个错误。

修改代码以后, 按照如下步骤运行该类。

(1) 代码编写完成后, 在项目窗口中选中 BookBeanTest.java 节点, 单击鼠标右键, 在弹出的菜单中选择“Compile File”选项对其进行编译, 然后选择“Run File”选项运行测试文件。

(2) 此时在 NetBeans 中会出现输出窗口, 显示测试通过的情况, 如图 25-11 所示。



图 25-11 运行测试的结果窗口

(3) 从图中可以看到“testBookName”测试没有通过。

(4) 在此界面中按下“Output”按钮, 测试结果窗口中会显示各个测试方法运行时输出到控制台的内容, 如图 25-12 所示。

25.5.4 在 NetBeans 中开发测试套件

前面例子介绍了在 NetBeans 中如何对已经存在的类开发测试用例进行测试, 下面将介绍如何在 NetBeans 中开发测试套件。按照如下步骤完成该例。

(1) 打开上一个例子中创建的项目, 依次选择主菜单中的“File”/“New File...”选项, 在弹出的新建文件向导窗口中设置文件的“Categories”为“JUnit”, “File Types”为“Test Suite”, 如图 25-13 所示。

(2) 单击“Next”按钮进入新建测试套件详细设置窗口, 在窗口中将“Class Name”设置为“BookTestSuite”, 如图 25-14 所示。

(3) 单击“Finish”按钮, 完成测试套件的新建, 这时 NetBeans 已经自动生成了测试套件的代码框架。代码如下:

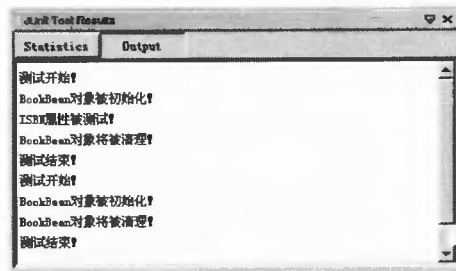


图 25-12 测试方法的运行输出

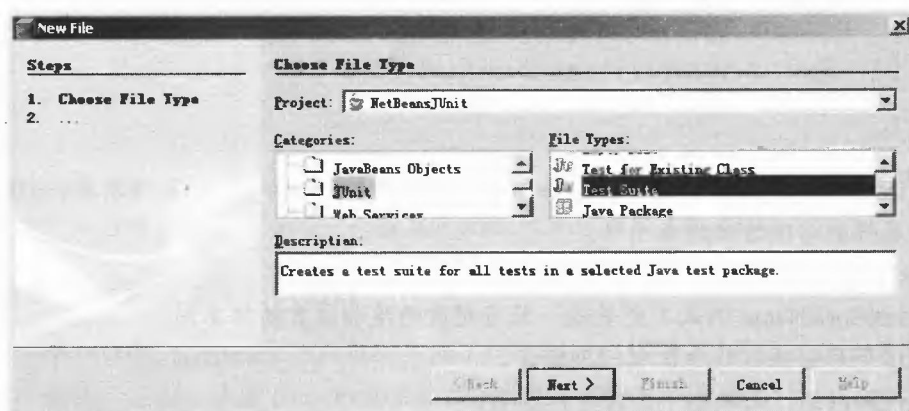


图 25-13 新建文件向导窗口

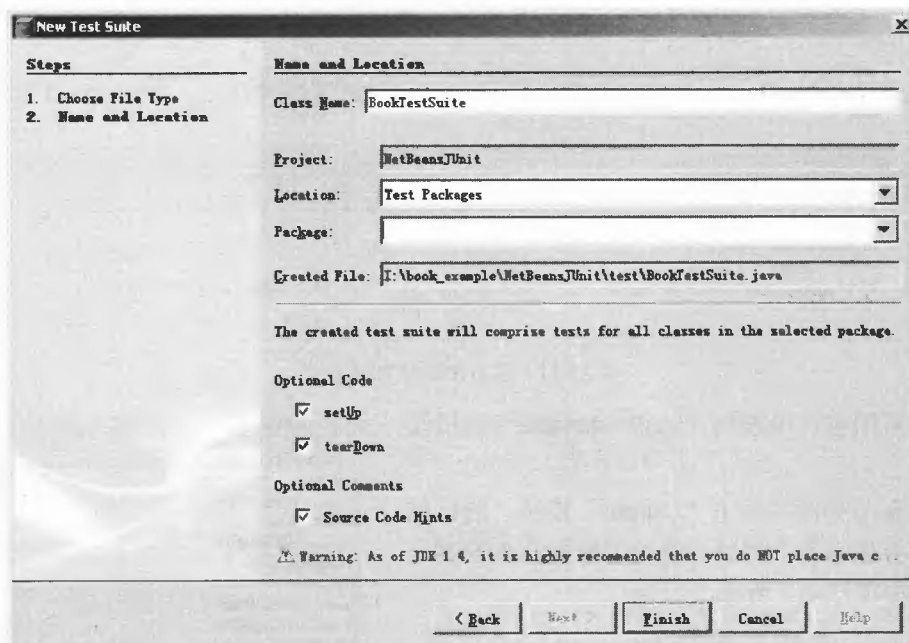


图 25-14 新建测试套件详细设置窗口

```
import junit.framework.*;
/*
 * BookTestSuite.java
 * JUnit based test
 *
 * Created on 2006年4月5日, 下午10:40
 */

/**
 *
 * @author Administrator
 */
```

```
public class BookTestSuite extends TestCase {

    public BookTestSuite(String testName) {
        super(testName);
    }

    protected void setUp() throws Exception {
    }

    protected void tearDown() throws Exception {
    }

    /**
     * suite method automatically generated by JUnit module
     */
    public static Test suite() {
        TestSuite suite = new TestSuite("BookTestSuite");
        suite.addTest(BookBeanTest.suite());
        return suite;
    }
}
```

- 读者可以发现，NetBeans 已经在测试套件的静态方法 suite 中，为测试套件中自动添加了在上个例子中开发的测试用例“BookBeanTest”中的测试。
- 如果需要加入系统没有自动添加的测试用例，只需要在 suite 方法中加入相应的代码即可。

(4) 在项目窗口中用鼠标右键单击 BookTestSuite.java 节点，选择右键菜单中的“Compile File”选项对其进行编译，然后选择“Run File”选项运行测试文件。

(5) 这时系统会出现测试运行的结果窗口，如图 25-15 所示。

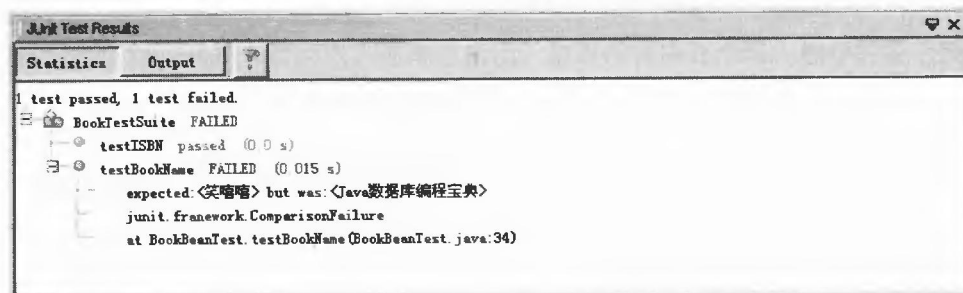


图 25-15 运行测试套件的结果

从图中可以看出，本例和上个例子一样运行了两个测试，有一个没有通过，在测试窗口中还列出了测试没有通过的原因。

在本节中，介绍了如何在 NetBeans 中开发测试用例和测试套件，从两个例子中可以看出使用 NetBeans 开发测试程序是非常方便快捷的。

25.6 小结

测试是开发过程中的一个非常重要的环节，良好的测试是保证软件质量的基石。本章主要对 NetBeans 平台开发进行了简单的介绍，包括单元测试、JUnit 简介、TestCase 类简介、TestSuite 类简介以及在 NetBeans 中通过 JUnit 进行测试。在下一章中，将对 NetBeans Profiler 进行简单的介绍。

第 26 章

NetBeans Profiler: 监控应用程序的执行

NetBeans Profiler 是 NetBeans 中一款非常有用的插件，可以监控应用程序的执行，给开发人员提供应用程序执行时的很多信息，本章将对 NetBeans Profiler 的安装和使用进行介绍。

26.1 NetBeans Profiler 简介

NetBeans Profiler 是由 NetBeans 组织开发的，一个具有完整功能的程序监控和分析工具，同时也是 NetBeans 的一个可选插件。主要的监控功能包括：

- CPU 监控；
- 内存使用情况监控；
- 线程运行情况监控。

在程序运行过程当中，通过 NetBeans Profiler 可以得知每个方法所耗费的 CPU 时间、内存堆使用的情况、垃圾收集的统计、线程的数量及运行状态，以及程序运行过程中对象创建的情况等。

通过这些信息，开发人员可以方便地分析出，如有没有内存泄漏、程序的各个线程设计是否合理高效等，为开发出更好的效能更高的程序提供了极大的方便。

26.2 NetBeans Profiler 安装配置

在安装 NetBeans Profiler 前首先可以从 Internet 上获取它的安装包（同样是免费的）。读者可以到 NetBeans 的官方网站：<http://www.netbeans.org> 下载。在获取了安装包后按照如下步骤进行安装。

(1) 启动安装程序 `netbeans-profiler-5_0-win.exe`，这时会看到安装向导界面，如图 26-1 所示。

(2) 单击“Next”按钮，进入版权声明界面，选择“I accept the items in the license agreement”选项。

(3) 单击“Next”按钮，进入 NetBeans 选择界面，如图 26-2 所示。

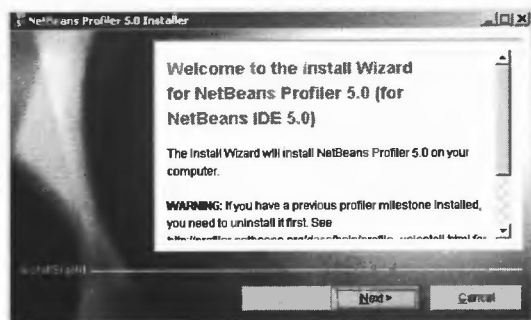


图 26-1 NetBeans Profiler 安装欢迎界面

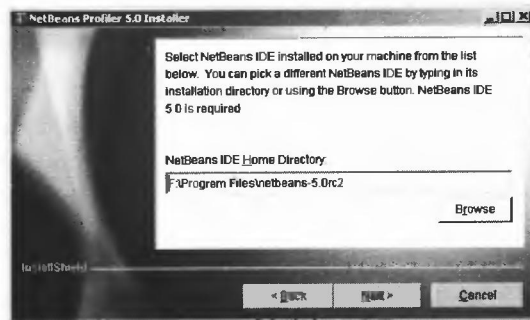


图 26-2 NetBeans 选择界面

(4) 单击“Browse”按钮，选择要安装 NetBeans Profiler 插件的 NetBeans 安装目录，然后连续单击“Next”按钮进行安装。

(5) 安装完毕后，在弹出的窗口中单击“Finish”按钮完成安装。

(6) 启动 NetBeans，如果主菜单中出现了“Profile”选项，说明 NetBeans Profiler 已经安装成功，如图 26-3 所示。

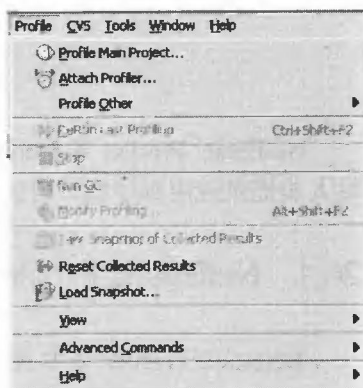


图 26-3 主菜单中的“Profile”选项

在本节中，将介绍使用 NetBeans Profiler 的方法，对 Java SE 应用中的线程状态进行监控。这可以帮助开发人员分析应用程序中的性能问题。本节将介绍如何在 NetBeans 中监控 Swing 应用中的线程状态，主要包括如下内容：

- 运行示例项目；
- 监控线程状态。

26.3.1 运行示例项目

在进行分析的过程中要用到一个例子程序，该程序已经包含在本书赠送的光盘中。本节将运行该示例项目。按照如下步骤，使用 NetBeans Profiler 监控应用程序中的各个线程。

(1) 依次选择主菜单中的“File”/“Open Project...”选项，打开本书所赠送光盘中第 26 章的项目“ProfilerSwing”。

(2) 在项目窗口中用鼠标右键单击“ProfilerSwing”节点，选择右键菜单中的“Clean and Build Project”选项，然后选择右键菜单中的“Run Project.”选项运行项目，将出现程序的界面，如图 26-4 所示。

(3) 把“计算的秒数”设置为 30，单击“启动”按钮，接着单击“关闭窗口体”按钮，这时会发现“关闭窗口体”按钮不起作用。

(4) 如果此时用其他程序的窗口覆盖此程序窗口，然后再把其他程序窗口移开，会发现此应用程序窗口的重绘也不能正常响应，如图 26-5 所示。

(5) 当 30s 过后, 程序会弹出信息提示对话框, 如图 26-6 所示, 按下对话框中的“确定”按钮, 对话框关闭、程序回到主窗体。

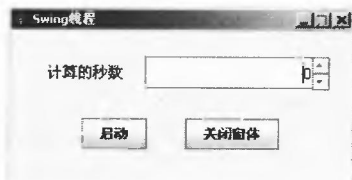


图 26-4 测试程序的界面



图 26-5 不能正常响应的程序窗体

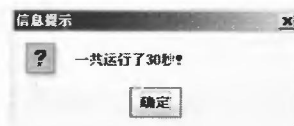


图 26-6 信息提示对话框

(6) 这时再单击“关闭窗体”按钮, 主窗体可以正常关闭。
完成了上述步骤, 就已经完成了示例项目的运行。

26.3.2 监控线程状态

上个小节中运行了示例程序, 本节将给出如何使用 NetBeans Profile 监控该线程状态, 按照如下步骤完成。

(1) 依次选择主菜单中的“Profile”/“Profile Main Project”选项, 用 NetBeans Profiler 来分析此应用程序。

(2) 这时系统会弹出一个对话框, 要求同意对项目的创建脚本进行修改, 如图 26-7 所示。

(3) 按下“OK”按钮同意修改, 这时系统会弹出选择任务类型窗口, 如图 26-8 所示。

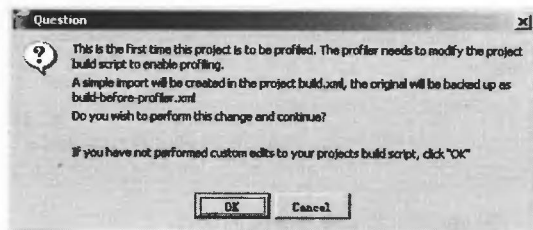


图 26-7 询问对话框

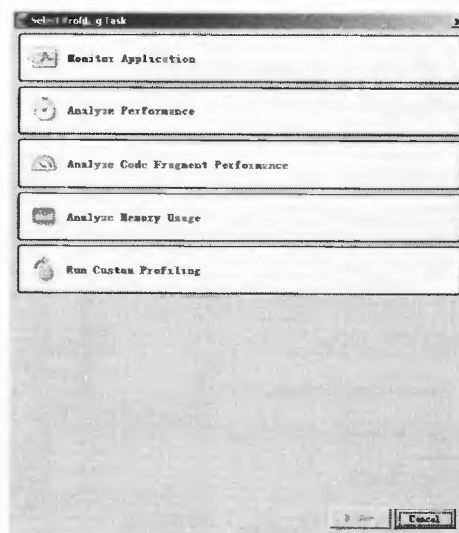


图 26-8 选择任务窗口

(4) 选择其中的“Monitor Application”选项, 这时“Monitor Application”选项会自动展开, 选中其中的“Enable Threads Monitoring”选项, 单击“Run”按钮。

(5) 如果是第一次使用 NetBeans Profiler, 系统会弹出对话框要求进行标准信息收集, 如图 26-9 所示。

(6) 按照提示, 依次选择主菜单中的“Profile”/“Advanced Commands”/“Run Profiler

Calibration”选项，进行标准信息收集，等到信息收集完毕后，系统会弹出如图 26-10 所示的对话框。

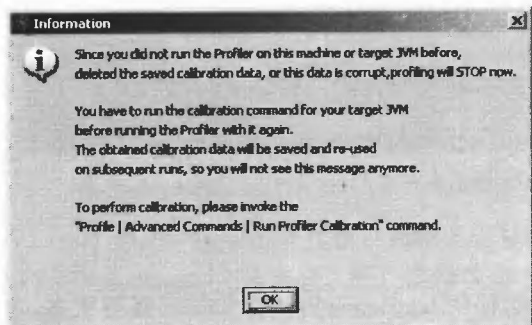


图 26-9 提示对话框

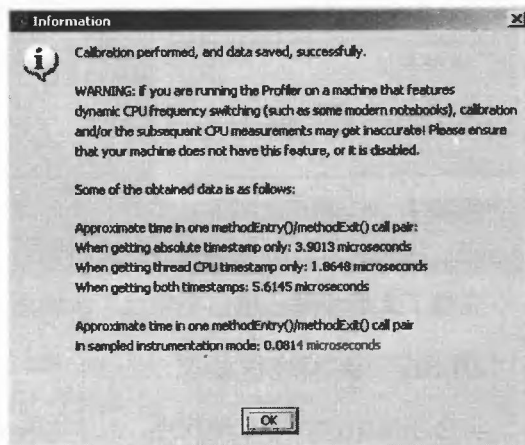


图 26-10 信息收集完毕对话框

(7) 这时再重复第(1)步~第(4)步，NetBeans Profiler 就开始工作了。可以看到 NetBeans 中出现了一个“Profiler”控制面板，如图 26-11 所示。

(8) 同时在 NetBeans 中，还出现显示程序中各个运行线程状态的窗口，如图 26-12 所示。

说明 在图中不同的颜色表示线程的不同运行状态。

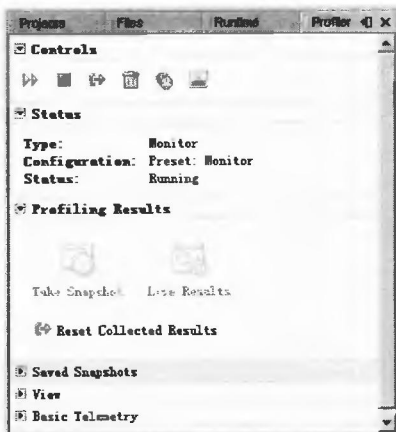


图 26-11 “Profiler”控制面板

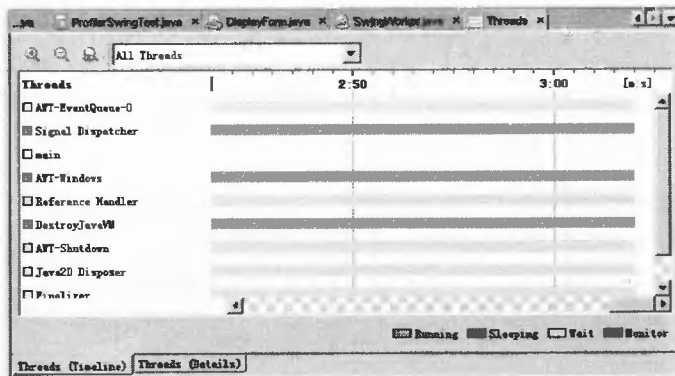


图 26-12 显示线程状态的窗口

- 绿色表示线程正在运行。
- 紫色表示线程由于调用了 Thread.sleep()方法正在睡眠。
- 黄色表示线程由于调用了 Object.wait()方法正在等待。
- 红色表示线程由于试图进入同步代码块或方法被阻塞。

(9) 选中应用程序窗口，设置秒数为 30s，单击“启动”按钮，这时从线程运行状态窗口中可以看到启动程序时不能响应其他界面事件的原因，如图 26-13 所示。

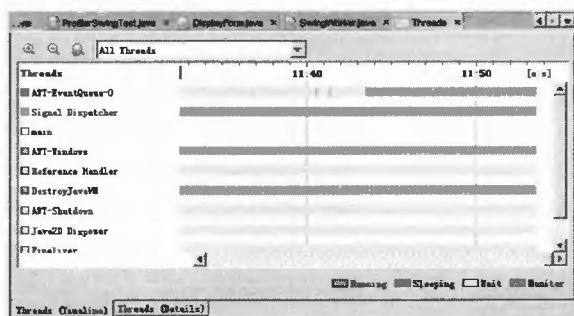


图 26-13 显示线程状态的窗口

说明

从图中可以看到，一旦启动程序，用来进行事件分发的线程 AWT-EventQueue-0 就处于满负荷运行中，故而不能正确响应其他事件。这种情况往往表示开发人员直接在事件处理方法中直接编写了业务量非常大的代码，这是一种不好的习惯，在正常情况下应该单独编写一个线程来执行大业务量的方法，在事件处理方法中完成的任务仅仅是启动线程，这样就不会出现程序“死”的现象了。

(10) 过 30s 后，关闭程序，在项目窗口中依次选择“ProfilerSwing”/“Source Packages”/“org.netbeans.profiler.profilerswing”/“ProfilerSwingTest.java”/“ProfilerSwingTest”/“Methods”/“jButton1ActionPerformed”节点，双击“jButton1ActionPerformed”节点，这时光标会出现在代码编辑器窗口，并定位到 jButton1ActionPerformed 方法中。

(11) 把 89~96 行之间的代码注释掉，同时取消 98~111 行代码的注释，然后重新编译和创建程序。

(12) 重复第 (7) 步~第 (11) 步，在弹出的程序主界面中设置秒数为 30s，单击“启动”按钮，这时会发现如果接着单击“关闭”按钮程序会立即关闭，不会出现停止的现象。

(13) 从线程状态窗口中也可以看到原因，如图 26-14 所示。

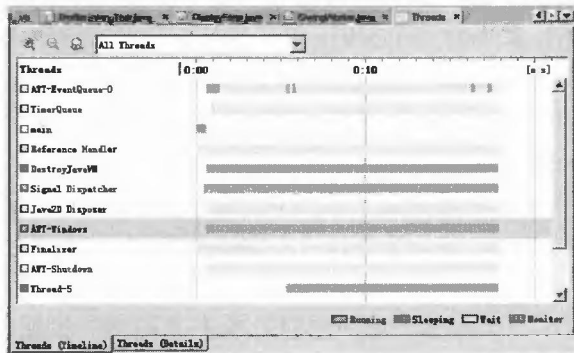


图 26-14 显示线程状态的窗口

- 从图中可以看出，由于开启了新的线程来完成大业务量的代码，这样不会占用太多的事件分发线程，所以在业务处理的过程中，事件分发线程可以很好地处理其他事件。
- 在图最下面的“Thread 5”线程是单独运行完成大业务量代码的。

(14) 选择线程状态窗口中的“Threads (Details)”页，可以查看选中线程的详细运行情况，如图 26-15 所示。

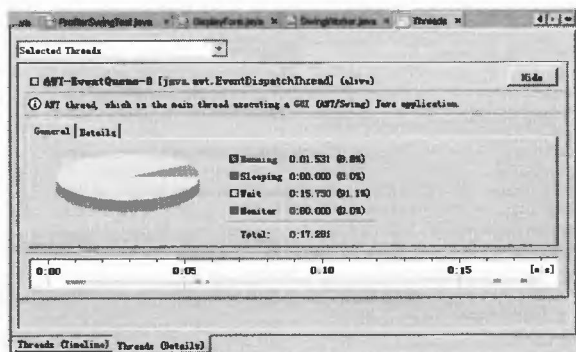


图 26-15 线程的详细运行情况

- 通过这个界面，开发人员可以分析是否每个线程占用了合理的执行时间。
- 本图中显示了事件分发线程大部分时间在等待，这是正常的。

本节通过一个例子，讲解了如何使用 NetBeans Profiler 来启动一个程序，以及监控程序执行时各个线程的执行状态，从而分析程序的执行效能。在下面的章节中将介绍 NetBeans Profiler 的其他功能。

26.4 监控指定方法的 CPU 使用情况

在实际应用中，CPU 的使用效能是开发人员非常关心的问题之一。例如，在检索系统中，可能一个检索比另一个检索运行得要慢一些，但是原因很难知道。在这种情况下就可以用 NetBeans Profiler 来监控应用程序中方法的 CPU 使用时间，这样对于找出问题的症结所在非常有帮助。在本节中将对这方面的知识进行介绍。

26.4.1 运行示例程序

在进行分析的过程中，要用到一个实例，该实例的源程序已经包含在本书赠送的光盘中。按照如下步骤对应用程序中方法的 CPU 使用时间进行监控即可。

(1) 依次选择主菜单中的“File”/“Open Project...”选项，打开本书所赠光盘中第 26 章的项目“ProfilerMethod”。

(2) 在项目窗口中用鼠标右键单击 ProfilerMethod 节点，选择右键菜单中的“Clean and Build Project”选项，然后选择“Run Project”选项运行项目，将出现如图 26-16 所示的程序界面。

(3) 在文本框中输入一个整数，如 91234，单击“提交”按钮，程序界面如图 26-17 所示。



图 26-16 程序运行界面

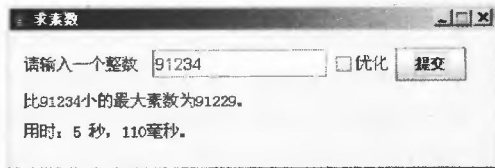


图 26-17 采用非优化方法的运行结果

(4) 重新输入一个整数 99999, 再选中“优化”复选框, 单击“提交”按钮, 程序界面如图 26-18 所示。

细心的读者应该发现, 优化方法的执行时间少于一个毫秒, 而非优化方法执行时间的长达“5 秒, 110 毫秒”, 两种方法差距相当大。

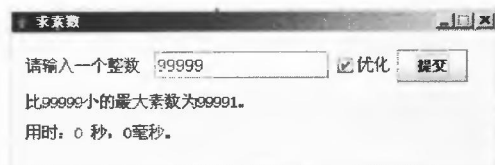


图 26-18 采用优化方法的运行结果

26.4.2 选择测试的根方法

下面将使用 NetBeans Profiler 对应用程序中方法的 CPU 使用时间进行监控, 以找到问题的瓶颈。本节用来选择测试的根方法, 按照如下步骤完成。

(1) 关闭求素数应用程序后, 依次选择主菜单中的“Profile”/“Profile Main Project”选项, 用 NetBeans Profiler 来分析此应用程序。

(2) 如果是第一次对项目进行测试, 系统会弹出一个对话框, 要求同意对项目的创建脚本进行修改, 如图 26-19 所示。

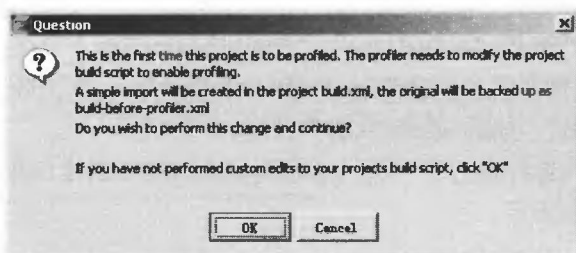


图 26-19 询问对话框

(3) 单击“OK”按钮同意修改, 这时系统会弹出选择任务类型窗口, 如图 26-20 所示。

(4) 在任务窗口中选择“Analyze Performance”选项, 系统将展开任务设置界面, 如图 26-21 所示。

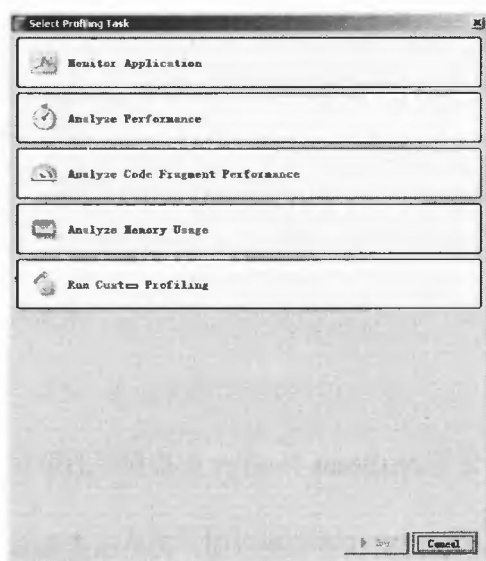


图 26-20 选择任务窗口

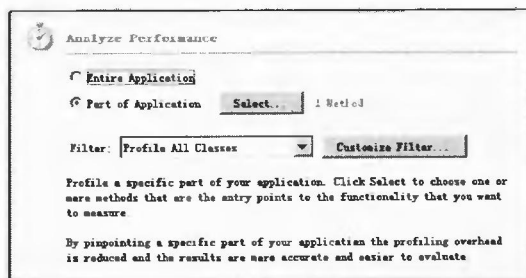


图 26-21 任务设置界面

(5) 选中“Part of Application”选项，然后单击“Select...”按钮选择测试的“根方法”，这时系统将弹出选择根方法界面，如图 26-22 所示。

(6) 单击“Add From Project...”按钮，系统将弹出方法选择界面，如图 26-23 所示。

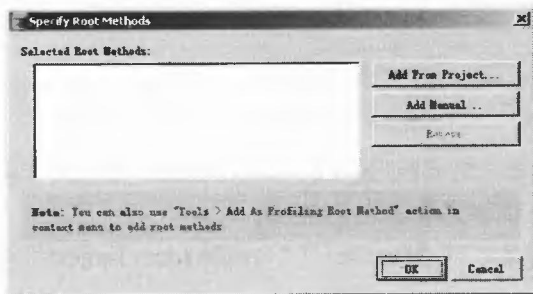


图 26-22 选择根方法界面

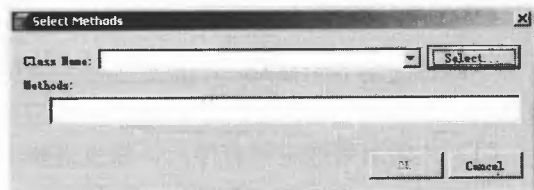


图 26-23 方法选择界面

(7) 单击“Select”按钮，系统将弹出类选择界面，如图 26-24 所示。

(8) 在类选择界面中展开 ProfilerMethod 节点，依次选中“Source Packages”/“oeg.netbeans.profiler.profilermethod”/“ProfilerMethod.java”/“ProfilerMethodTest”节点，然后单击“OK”按钮。

(9) 这时在方法选择界面中，会列出 ProfilerMethodTest 类下的所有方法，选中其中的“jButtonActionPerformed”方法，单击“OK”按钮。

(10) 此时系统回到选择根方法界面，这时列表中出现了指定的根方法“jButtonActionPerformed”，如图 26-25 所示。

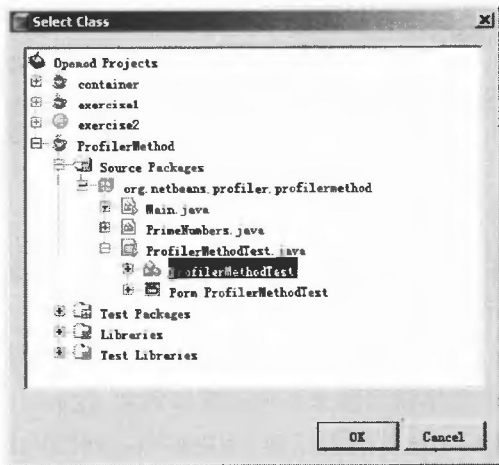


图 26-24 类选择界面

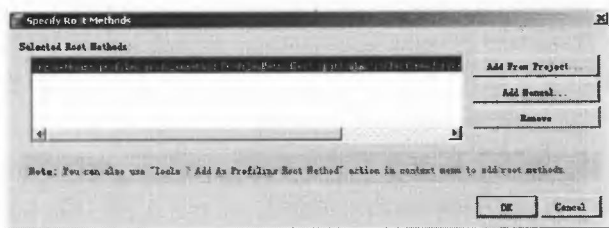



图 26-25 选择根方法界面

完成了上述步骤，就已经完成了根方法的选择。

26.4.3 监控程序的运行

在上一个小节中完成了根方法的选择，本节将使用 NetBeans Profiler 监控程序的运行，可按照如下的步骤进行操作。

(1) 在如图 26-25 所示的窗口中，单击“OK”按钮回到任务设置界面，再单击界面右下方的  按钮开始运行被测试的程序。

(2) 程序开始运行后, 出现如图 26-16 所示的界面。而 NetBeans Profiler 已经在后台运行。同时可以看到 NetBeans 中出现了一个“Profiler”控制面板, 与图 26-11 相同。

(3) 在程序运行界面中输入“91234”, 不选中优化复选框, 单击“提交”按钮开始运行。由于性能问题, 程序需要经过一段时间程序才出现结果, 与图 26-17 基本相同。

(4) 单击“Profiler”控制面板中的“Take Snapshot”选项, 在 NetBeans 中, 将出现程序运行时调用的各个方法的 CPU 使用时间统计列表。选中列表中的“Combined”页, 界面如图 26-26 所示。

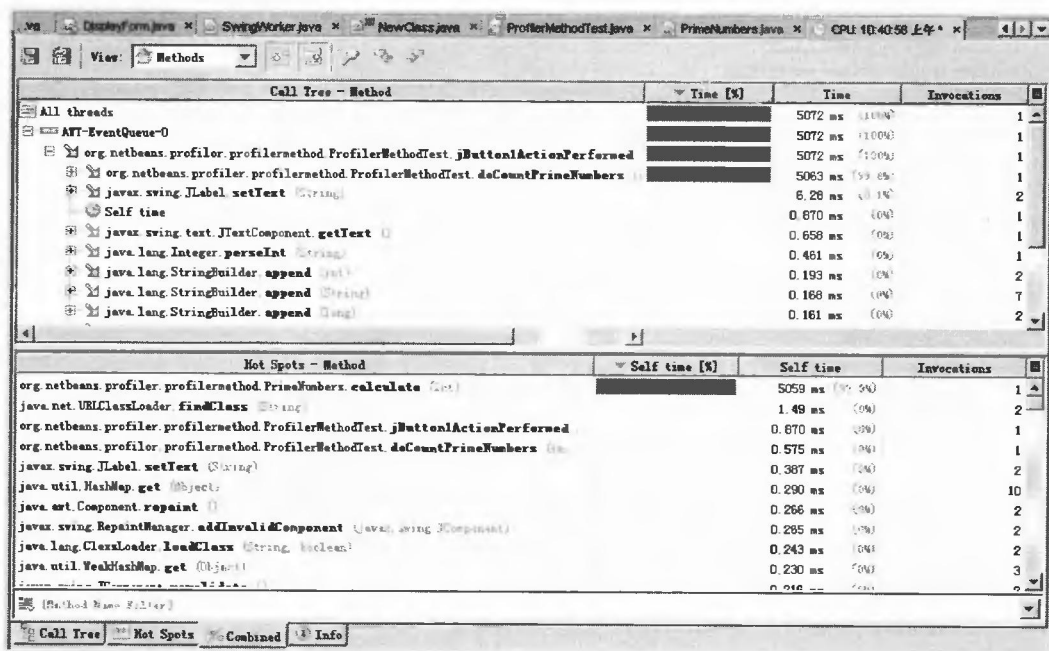


图 26-26 未采用优化算法方法占用 CPU 时间统计结果

- 在界面上方列出的是在程序执行过程中, 从根方法开始的级联调用树, 同时也给出了每个方法的执行总时间, 以及占总时间的百分比。
- 在界面上方列出的是在程序执行过程中, 每个方法实际自己运行的时间, 以及占总运行时间的百分比, 并且按百分比从大到小排序。
- 从图中可以看到, calculate 方法占用了最多的时间, 这个方法是用来求指定数下最大素数的, 其采用了没有优化的算法。

提示

方法执行总时间包括其调用其他方法的时间, 而方法实际自己运行的时间不包括对其他方法调用所占的时间。

(5) 依次选择主菜单中的“Profile”/“Reset Collected Results”选项, 清空统计数据。

(6) 在程序运行界面中选中“优化”复选框, 单击“提交”按钮执行。由于采用优化算法, 程序运行时间很短, 与图 26-18 基本相同。

提示

读者可能会发现采用 NetBeans Profiler 后程序执行比使用之间略慢。这是正常现象, 因为 NetBeans Profiler 进行统计也需要执行时间。

(7) 重复步骤(4), 这时的界面如图 26-27 所示。

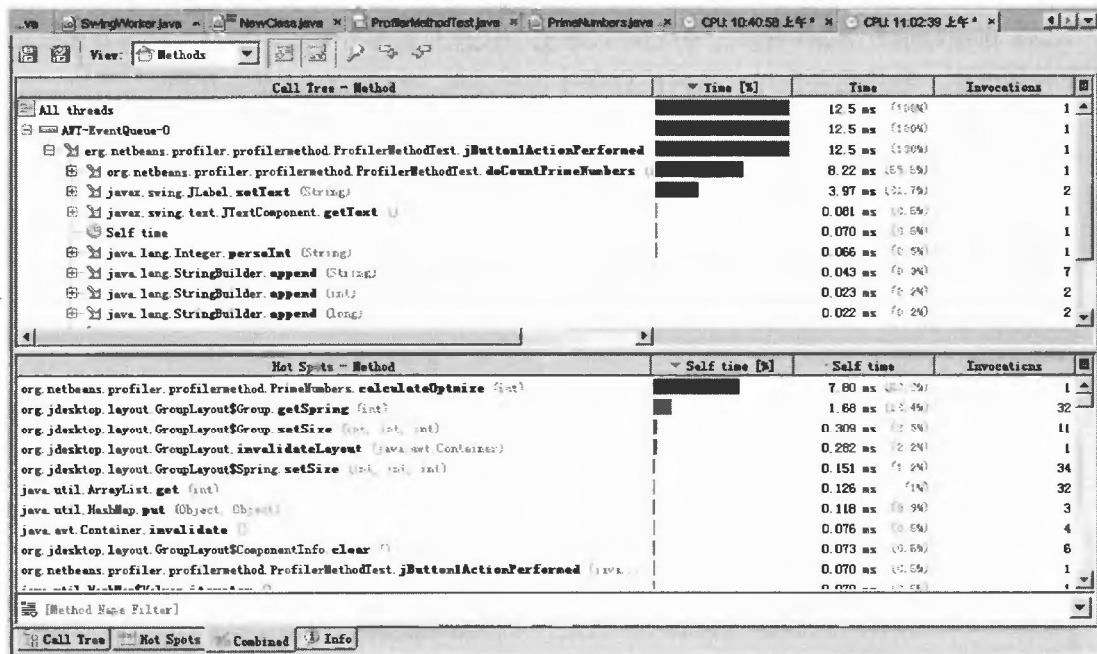


图 26-27 采用优化算法方法占用 CPU 时间统计结果

- 在本界面中, 显示方法执行的总时间比非优化算法低。在采用优化算法后, 计算相应素数的方法执行时间只占 62%, 而未采用优化算法时占了将近 100%。
- 通过分析各个方法自己所占总执行时间的比例, 可以分析出程序执行性能的瓶颈, 以及各个方法执行时间所占比例是否合理。

本节通过一个例子, 讲解了如何使用 NetBeans Profiler 来启动一个程序, 以及监控指定方法的 CPU 运行时间, 从而分析程序的执行效能瓶颈, 以及每个方法所占 CPU 执行时间的比例是否相同。在下面的章节中, 将介绍如何使用 NetBeans Profiler 发现内存漏洞。

26.5 通过 NetBeans Profiler 寻找内存漏洞

在前面的小节中, 介绍了如何使用 NetBeans Profiler 监控应用程序中线程的使用状态, 以及 CPU 使用的情况。在本小节将介绍如何通过 NetBeans Profiler 寻找内存漏洞, 这对于开发人员来说是一个非常实用的功能。

在进行分析的过程中要使用到一个例子程序。这个程序已经包含在本书赠送的光盘中。按照如下步骤就可以寻找内存漏洞。

(1) 依次选择主菜单中的“File”/“Open Project...”选项, 打开本书所赠送光盘中第 26 章的项目“ProfileMemoryLeaking”。

(2) 在项目窗口中用鼠标右键单击 ProfileMemoryLeaking 节点, 选择右键菜单中的“Clean and Build Project”选项, 然后选择“Run Project.”选项运行项目, 将出现如图 26-28 所示的程序界面。

- 单击“开始泄漏”按钮，应用程序中的线程将不断创建不可回收的对象，这些对象不断占用内存空间，造成内存泄漏。
- 单击“停止泄漏”按钮，应用程序停止内存泄漏。



图 26-28 内存泄漏例程界面

(3) 关闭应用程序，然后依次选择主菜单中的“Profile”/“Profile Main Project”选项，用 NetBeans Profiler 来分析此应用程序。

(4) 如果是第一次对项目进行测试，系统会弹出一个对话框，要求同意对项目的创建脚本进行修改，如图 26-19 所示。单击“OK”按钮同意修改，这时系统会弹出选择任务类型窗口，如图 26-20 所示。

(5) 在选择任务类型窗口中选择“Monitor Application”选项，再单击界面右下方的 按钮，开始运行被测试的程序。

(6) 这时可以看到，NetBeans 中出现了一个“Profiler”控制面板，与图 26-11 相同。单击“Profiler”控制面板中的 按钮，这时在 NetBeans 中，出现“VM Telemetry Overview”界面，如图 26-29 所示。

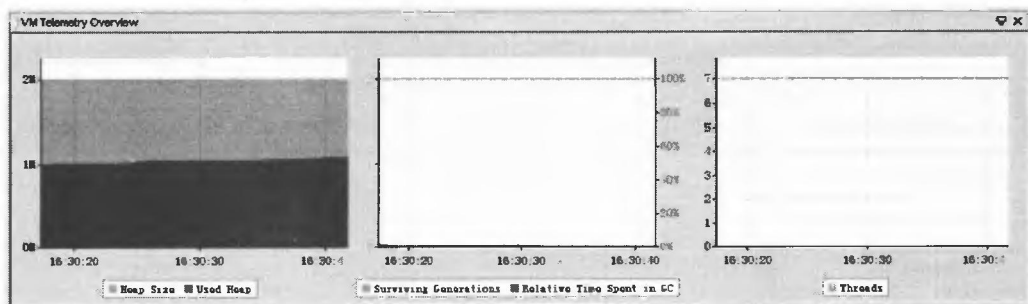


图 26-29 虚拟机运行情况监测界面

- 在图中最左边的是虚拟机内存使用情况，蓝色表示已经使用的堆内存，红色表示未被使用的堆内存。
- 中间的部分是垃圾收集情况，以及对象代数的情况。红色线条表示目前虚拟机中所有非垃圾对象的代数，蓝色线条表示垃圾收集线程运行的情况。
- 右边的图表示的是虚拟机中所有未死亡的线程数量。

提示

对象的代数是指在某个时刻虚拟机中，所有非垃圾对象具有不同年龄的数量，一个对象的年龄是指其从诞生到这个时刻的时间跨度。NetBeans Profiler 可以对 CPU 性能、内存的使用情况进行更详细的分析，但不可以同时进行。为了实现对垃圾收集和堆使用情况的详细监控，需要修改 Profiler 的设置。

(7) 依次选择主菜单中的“Profile”/“Modify Profiling...”选项，在弹出的界面中选择“Analyze Memory Usage”，将出现内存分析设置界面，如图 26-30 所示。

(8) 选中“Record both object creation and garbage collection”单选按钮，选中“Record Stack Trace for Allocations”复选框，单击“OK”按钮，开始详细分析。

(9) 这时单击应用程序界面中的“开始泄漏”按钮，开始内存泄漏。

(10) 这时在虚拟机运行情况监测界面中，中间显示垃圾收集和对象代数的图，如图 26-31

所示。

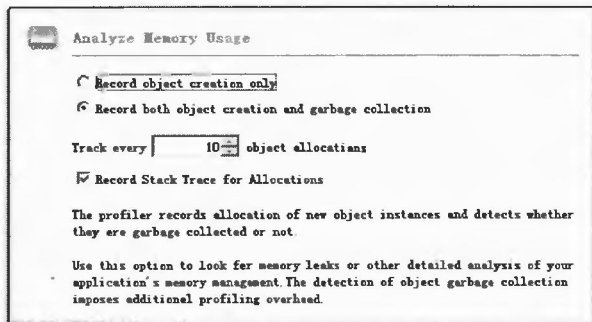


图 26-30 内存分析设置界面

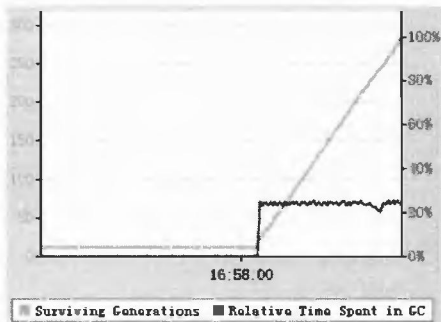


图 26-31 垃圾收集和对象代数监控界面

- 这时，从图上可以看出，垃圾收集工作在不断进行，但对象代数依然持续增加，这种情况往往意味着存在内存泄漏的问题。

(11) 依次选择主菜单中的“Profile”/“View”/“Live Results”选项，系统将显示一个动态更新系统中的所有类型对象的情况列表，如图 26-32 所示。

Class Name - Live Allocated Objects	Live Bytes	Live Objects	Allocated Objects	Avg. Age	Generations
int[]	3,484 B (7.3%)	16 (1.8%)	30	72.6	14
java.util.HashMap\$Entry	2,952 B (7.1%)	123 (12.7%)	173	120.8	9
java.lang.String	792 B (1.9%)	33 (3.5%)	138	119.1	9
char[]	2,088 B (5.3%)	36 (3.9%)	224	115.4	8
java.util.HashMap	400 B (1.0%)	10 (1.1%)	13	122.4	7
java.lang.Object[]	1,520 B (3.8%)	18 (1.9%)	234	94.8	6
java.util.HashMap\$Entry[]	1,232 B (3.1%)	9 (0.9%)	16	122.2	6
java.lang.Integer	672 B (1.6%)	42 (4.5%)	43	122.9	6
java.util.Hashtable\$Entry	2,040 B (5.1%)	85 (8.9%)	92	122.2	5
java.lang.ref.SoftReference	320 B (0.8%)	10 (1.1%)	16	120.5	4
java.lang.Object	40 B (0.1%)	5 (0.5%)	7	122.2	4
java.lang.String[]	400 B (1.0%)	9 (1.0%)	10	122.7	3
java.lang.ref.WeakReference	192 B (0.5%)	8 (0.8%)	33	44.8	3
java.lang.reflect.Constructor	192 B (0.5%)	3 (0.3%)	3	122.3	3
java.util.Hashtable	120 B (0.3%)	3 (0.3%)	4	122.7	3
java.util.Vector	72 B (0.2%)	3 (0.3%)	8	80.3	3

图 26-32 系统中所有类型对象的情况列表

- Allocated Objects 列表示 NetBeans Profiler 监视到的对象数量，例如在本例中此时 int[] 对象的总数量为 30。
- Live Objects 表示此时还没有成为垃圾被收集的此类对象的数量。
- 两列 Live Bytes 分别用图形和文本的形式，表示此类对象所占堆空间的大小。
- Avg. Age 列表示此类还没有成为垃圾被收集的对象的平均年龄。
- Generations 列表示此类还没有成为垃圾被收集的对象的代数。

(12) 随着程序的继续运行，NetBeans Profiler 将动态地更新列表中的显示。随着时间的推移，int[] 对象的代数不断增加。

(13) 依次选择主菜单中的“Profile”/“Take Snapshot of Collected Results”选项，在产生的列表中选中 int[] 行，单击鼠标右键，在弹出的菜单中选择“Show Allocation Stack Traces”选项，系统将显示所有产生 int[] 对象的方法的调用栈，如图 26-33 所示。

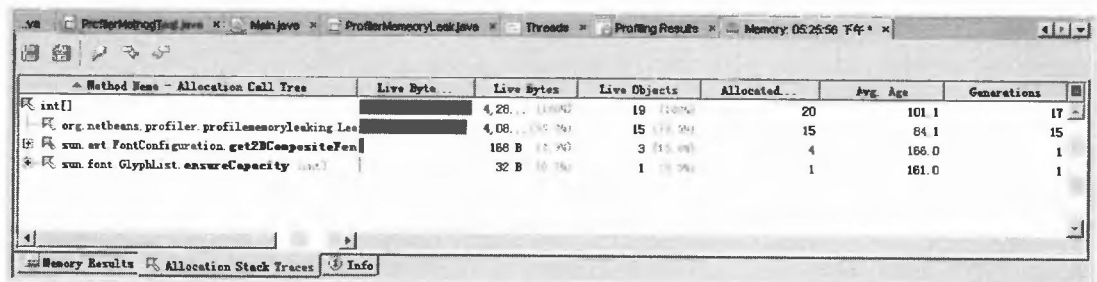
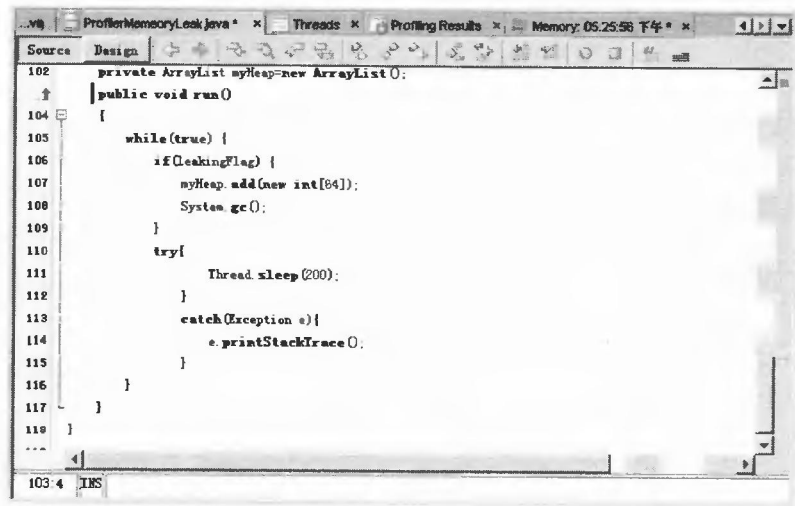


图 26-33 产生 int[] 对象的方法调用栈

从图中可以看出产生最多代数 int[] 对象的方法是 run() 方法。

(14) 在列表表中选中 run() 方法，单击鼠标右键，在弹出的菜单中选择 “Go To Source...” 选项，这时光标会出现在代码编辑器的 run() 方法上，如图 26-34 所示。



第 27 章

将 Eclipse 项目导入 NetBeans

Eclipse 也是一种当前比较流行的 Java 集成开发环境，现在很多项目都是通过 Eclipse 开发的。对于使用 NetBeans 的开发人员来说，如何将已有的 Eclipse 项目导入 NetBeans 中，进行再开发，是一件很重要的事情。本章将为读者介绍如何将已有的 Eclipse 项目导入 NetBeans 当中，主要包括如下内容：

- 将 Eclipse 项目导入到 NetBeans；
- 获取与安装 Eclipse 项目导入器；
- 导入 Eclipse 项目。

27.1 概述

对于在 Eclipse 中开发的项目，可以很容易地导入到 NetBeans 中。Eclipse 项目导入器可以自动完成 Eclipse 项目的导入工作，自动处理 Eclipse 项目的元数据，并将其直接映射到新建的 NetBeans 项目中。

NetBeans 5.x 可以对 Eclipse 项目（包括 Eclipse 3.0 兼容的项目）进行导入。开发人员可以在 NetBeans 的更新中心获取该模块。对于后续发布的 NetBeans 版本，Eclipse 项目导入器可能会成为标准 NetBeans 的一部分。

在将 Eclipse 项目导入 NetBeans 后，NetBeans Eclipse 项目导入器不仅能够识别、自动修正 Eclipse 项目与 NetBeans 项目的差异，还能自动识别错误的 Eclipse 项目引用，同时提供解决这些问题的正确操作。

NetBeans Eclipse 项目导入器的应用非常灵活，可以从 Eclipse 工作台选择项目。在 Eclipse 工作台中，对于有依赖关系的项目，可以自动识别并标注其依赖关系。另外，NetBeans Eclipse 项目可以在 NetBeans 中无须指定 Eclipse 工作台位置的情况下，将独立的 Eclipse 项目导入 NetBeans。

27.2 获取与安装 Eclipse 项目导入器

在上一节中，对将 Eclipse 项目导入 NetBeans 进行了简单的说明，本节将为读者介绍如

何获取与安装 Eclipse 项目导入器。NetBeans Eclipse 项目导入器是 NetBeans 的可选模块，因此不包含在标准的 NetBeans 中。读者可以按照如下步骤获取与安装 Eclipse 项目导入器。

(1) 启动 NetBeans，选择主菜单中的“Tools”/“Update Center”选项，出现如图 27-1 所示的窗口。在这里可以选择要连接到的更新中心。

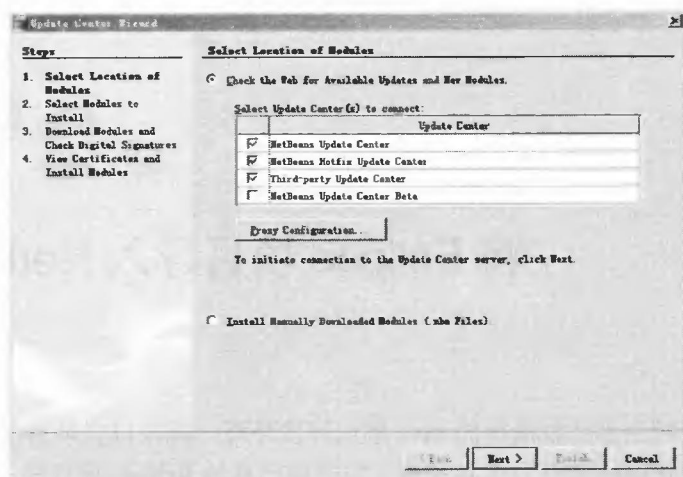


图 27-1 选择模块位置窗口

(2) 在此使用默认选项，单击“Next”按钮，经过一段时间的连接以后，出现如图 27-2 所示窗口。

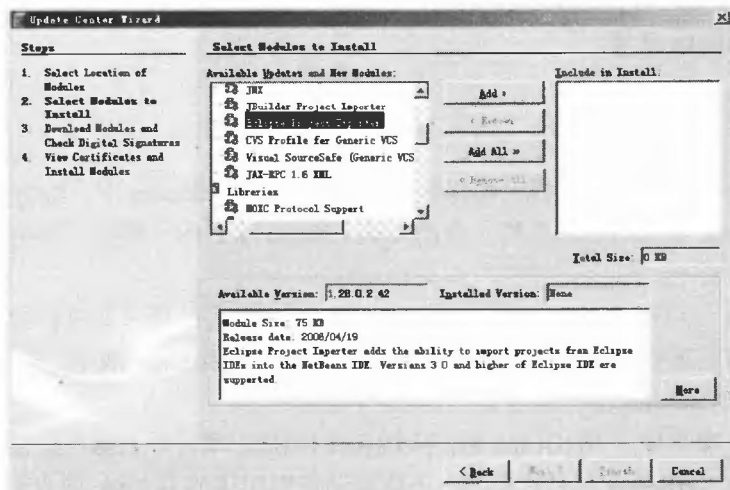


图 27-2 选择要更新的模块

(3) “Available Updates and New Modules”列表框列出了当前可以更新的模块。这里，选中“Eclipse Project Importer”选项，单击“Add”按钮，窗口如图 27-3 所示。

(4) 在“Include in Install”列表框中列出了当前选中模块中包含的可安装组件。如果不安装该组件，可以选中该组件，单击“Remove”按钮，从安装列表中移除该组件，在此使用默认设置。单击“Next”按钮，出现如图 27-4 所示窗口。这里给出了安装 Eclipse 项目导入器的安装许可证。

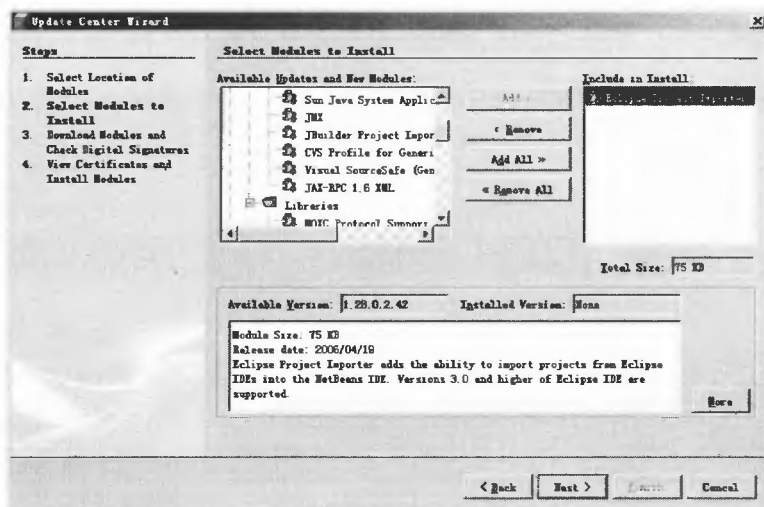


图 27-3 包含的可安装组件

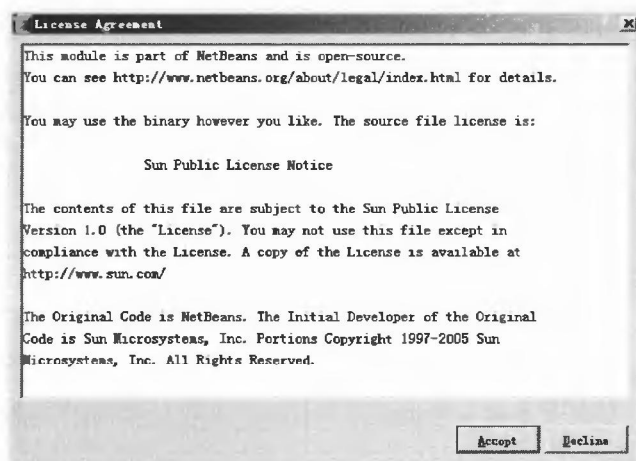


图 27-4 安装许可证窗口

(5) 单击“Accept”按钮，窗口如图 27-5 所示。该窗口显示了下载 Eclipse 项目导入器的进度。

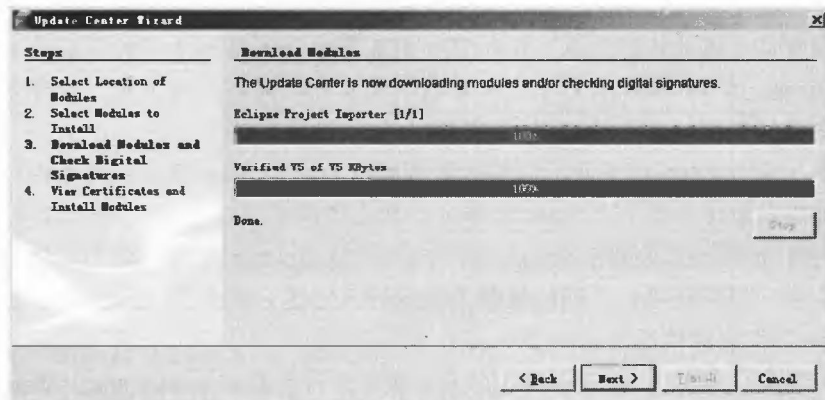


图 27-5 下载进度窗口

(6) 下载完成以后, 单击“Next”按钮, 此时窗口如图 27-6 所示。

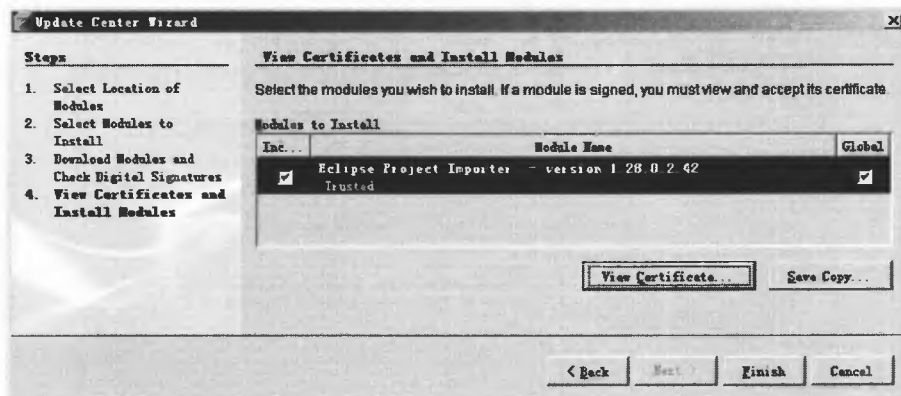


图 27-6 选择安装模块

(7) 这里列出了当前可供安装的模块, 分别选中“Inc...”与“Global”对应的复选框后, 单击“Finish”按钮即可完成。

完成了上述步骤, 就已经成功地完成了 Eclipse 项目导入器的获取与安装。

27.3 导入 Eclipse 项目

上一节中介绍了如何获取 Eclipse 项目导入器, 本节将为读者介绍如何将 Eclipse 项目导入到 NetBeans 当中, 主要包括如下内容:

- 两种导入方式的说明;
- 导入 Eclipse 项目并保存依赖关系;
- 导入 Eclipse 项目并忽略依赖关系。

27.3.1 两种导入方式的说明

NetBeans 的 Eclipse 项目导入器, 提供了两种将 Eclipse 项目导入到 NetBeans 的方式。

- 从 Eclipse 工作台中选择 Eclipse 项目。在 Eclipse 工作台中, NetBeans 的 Eclipse 项目导入器可以自动识别、标注项目之间的依赖关系, 进而选择项目。这种方式可以保存项目之间的依赖关系, 从而可以更容易地导入依赖于其他项目的 Eclipse 项目。NetBeans 的 Eclipse 项目还可以自动确定项目之间的依赖关系。这种方式也可以导入没有任何依赖关系的 Eclipse 项目。
- 只导入某个 Eclipse 项目, 而忽略该项目与其他 Eclipse 项目之间可能的依赖关系。这种方式最适合没有 Eclipse 工作台, 但是可以直接访问 Eclipse 项目的情况。

大多数的情况下, 应该使用 Eclipse 工作台来导入 Eclipse 项目, 因为这样可以保持项目之间的依赖关系。而忽略项目之间的依赖关系的导入方式, 则适于已经卸载了 Eclipse IDE、删除了 Eclipse 工作台的情况。

本章使用了一个名称为 Eclipse 的工作台, 该工作台中存在 book、print、Web 和 ShopCart 4 个项目。其中 print 中存在两个名称为 PrintHello 与 PrintOneToNine 的 Java 文件, 这两个文

件都用于在控制台中输出一些信息，PrintHello 用来在控制台中输出 “This Is Hello Class”，代码如下：

```
public class PrintHello {  
    public static void main(String[] args) {  
        System.out.println("This Is Hello Class ");  
    }  
}
```

PrintOneToNine 主要用来输出从 0~9 之间的数字，代码如下：

```
public class PrintOneToNine {  
    public static void main(String[] args) {  
        for(int i=0;i<10;i++){  
            System.out.println("....."+i+".....");  
        }  
    }  
}
```

27.3.2 导入 Eclipse 项目并保存项目依赖关系

在本小节中，将介绍从 Eclipse 工作台导入 Eclipse 项目的必要步骤。使用这种方式导入 Eclipse 项目时，项目导入器会自动确定项目之间的依赖关系，并自动导入与目标项目有关系的项目。

可以按照如下步骤完成 Eclipse 项目的导入。

(1) 在 NetBeans 的主菜单中选择 “File” / “Import Project” / “Eclipse Project” 选项，出现图 27-7 所示的窗口。

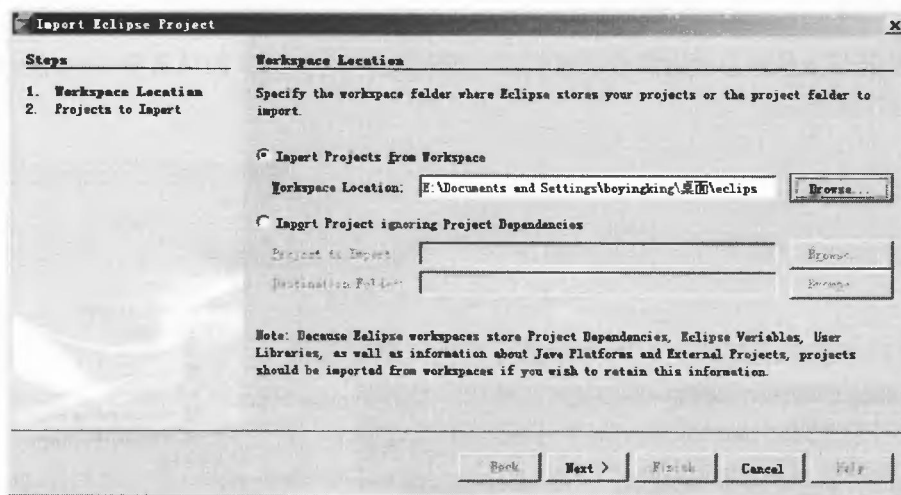


图 27-7 选择工作台窗口

(2) 选中 “Import Project from Workspace” 选项，单击 “Browse” 按钮，在弹出的文件选择对话框中，选择要导入的工作区。选择完成以后，单击 “Next” 按钮，出现如图 27-8 所示的窗口。

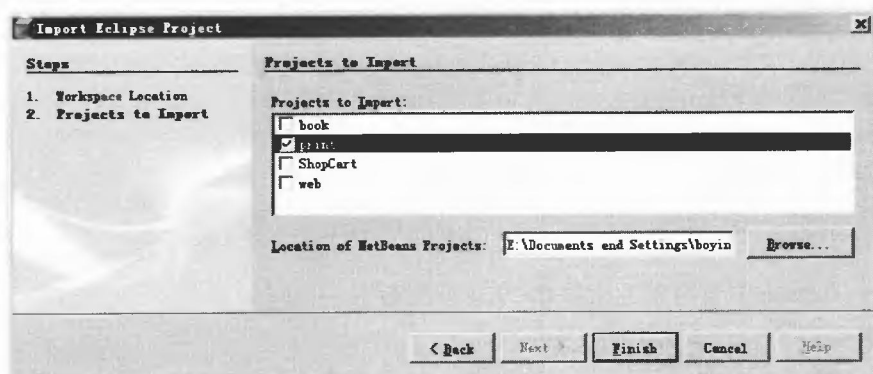


图 27-8 导入项目界面

(3) 在“Project to Import”列表框中，列出了希望导入的 Eclipse 项目。在“Location of NetBeans Projects”文本框中，列出了导入后所创建的 NetBeans 项目的位置，单击“Browse”按钮可以更改该位置。

如果导入的项目与其他项目存在着依赖关系，项目导入器会自动确定这种依赖关系，并且对同时导入的其他项目进行标注，此时主项目会被导入，而其他项目也会被自动标注为“导入”。

(4) 单击“Finish”按钮，完成 Eclipse 项目到 NetBeans 项目的导入过程。

如果 NetBeans 在处理 Eclipse 项目信息时，发现了 Eclipse 项目的差异，则会出现如图 27-9 所示的对话框。在如图 27-9 所示窗口中，显示了 Eclipse 项目的差异，以及 NetBeans 为解决这些差异所进行的操作。如果出现该对话框，只需要单击“OK”按钮即可，而不需要进行其他操作。

(5) 如果导入的项目没有其他依赖关系，则可以在 NetBeans 的项目窗口中查看导入的项目，如图 27-10 所示。

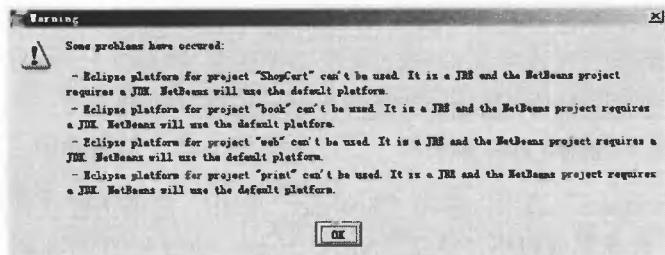


图 27-9 处理项目差异窗口

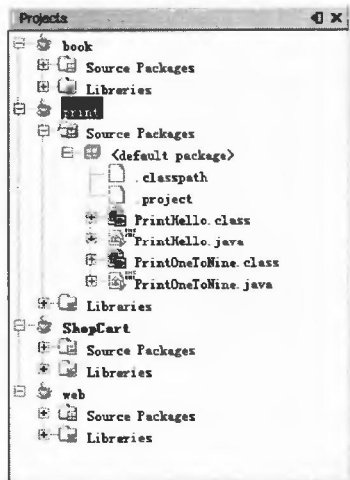


图 27-10 导入 Eclipse 项目后的项目窗口

(6) 在如图 27-10 所示窗口中用鼠标右键单击 PrintHello.java 节点, 选择右键菜单中的“Run File”选项运行该程序, 此时 NetBeans 的输出窗口中内容如图 27-11 所示。

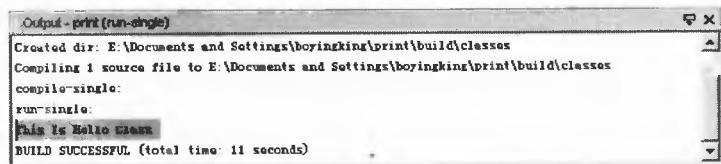


图 27-11 程序运行输出窗口

27.3.3 导入 Eclipse 项目并忽略依赖关系

在前面介绍了如何导入 Eclipse 项目并保存依赖关系。本节将为读者介绍当 Eclipse 工作台中不存在的时候, 或者希望忽略项目之间的依赖关系, 而导入特定项目的时候, 导入项目所需要的步骤。使用这种方法导入 Eclipse 项目的时候, NetBeans Eclipse 项目导入器会忽略项目之间可能存在的任何依赖关系。按照如下步骤完成项目的导入。

(1) 在 NetBeans 的主菜单中依次选择“File”/“Import Project”/“Eclipse Project”选项, 出现如图 27-12 所示的窗口。

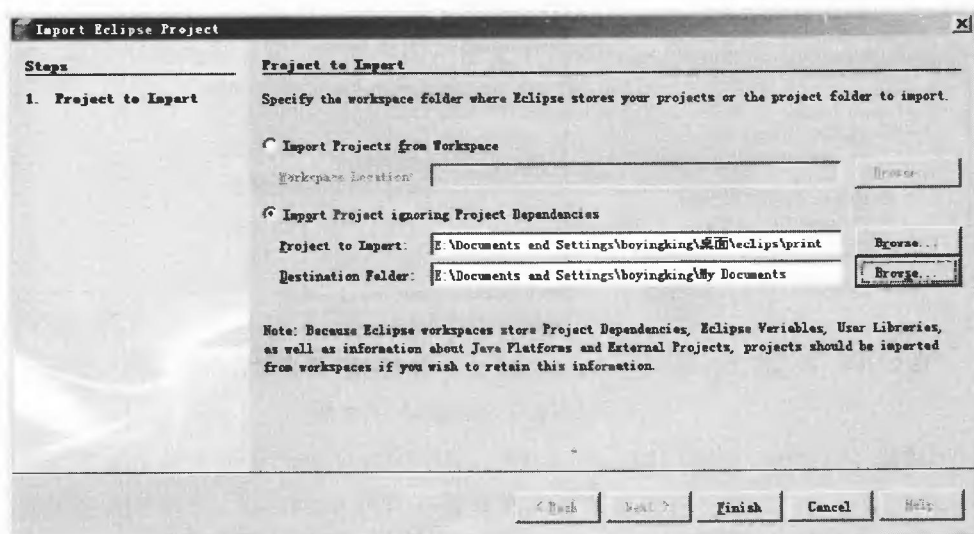


图 27-12 选择导入项目窗口

(2) 图 27-12 与图 27-7 所示的窗口内容看起来很相似, 不过在此选中的是“Import Project ignoring Project Dependencies”选项, 而非“Import Projects from Workspace”选项。

(3) 可以在图 27-12 所示窗口中设置两项内容, “Project to Import”文本框给出了要导入的 Eclipse 项目的位置; “Destination Folder”的值给出了用来存储导入项目的位置。设置完成以后, 单击“Finish”按钮完成项目的导入。

(4) 如果在处理 Eclipse 项目信息时, 发现了 Eclipse 项目的差异, 系统会自动处理并解决这些差异。此时 NetBeans Eclipse 项目导入器会出现一个警告对话框, 如图 27-13 所示。

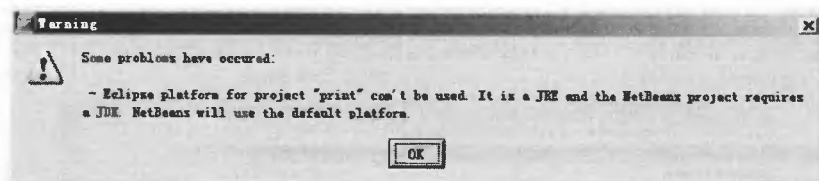


图 27-13 NetBeans 已经改正过的项目差异提醒

(5) 在如图 27-13 所示的窗口中显示了 Eclipse 项目的差异, 以及 NetBeansIDE 为解决这些差异所进行的操作。此时只需单击“OK”按钮即可。

提示

如果项目导入器没有在 Eclipse 项目中发现项目的引用问题, 只要没有其他需要导入的与该项目有依赖关系的项目, 那么便可以使用 NetBeans 来开发新的导入项目了。如果导入的项目没有其他的项目依赖关系, 则可以在 NetBeans 的项目窗口中查看导入的项目, 如图 27-14 所示。

(6) 在项目窗口中用鼠标右键单击 PrintOneToNine.java 节点, 选择右键菜单中的“Run File”选项运行该程序, 此时在 NetBeans 的输出窗口中会显示如图 27-15 所示内容。

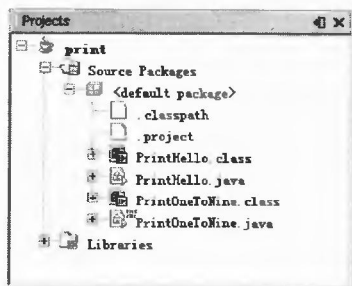


图 27-14 项目窗口内容

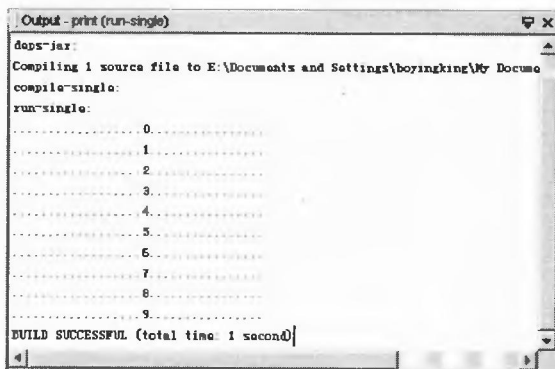


图 27-15 程序运行窗口

27.4 小结

Eclipse 也是一种当前比较流行的 Java 集成开发环境, 现存的很多项目, 都是通过 Eclipse 开发的。本章介绍了如何将 Eclipse 项目导入到 NetBeans, 包括导入 Eclipse 项目到 NetBeans 的简单说明、获取与安装 Eclipse 项目导入器以及如何导入 Eclipse 项目。

附录 A

NetBeans 5.0 的下载过程

在正文的章节中，已经详细地介绍了如何使用 NetBeans。在这里将为读者介绍如何获得 NetBeans 5.0 安装程序。具体过程如下。

(1) 打开浏览器，在地址栏输入 www.netbeans.org，进入 NetBeans 的官方网站，如图 A-1 所示。这个网站内容丰富、信息全面，是使用 NetBeans 的开发人员必去网站之一。

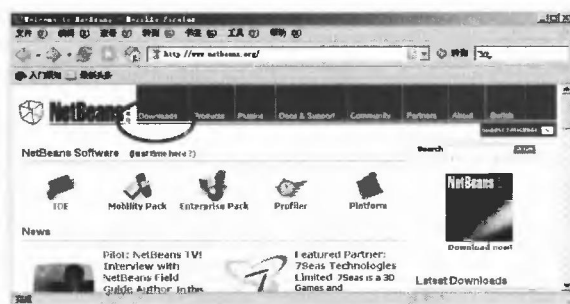


图 A-1 NetBeans 官方网站的页面

(2) 在图 A-1 的页面中，单击工具栏中的“Downloads”链接，进入 NetBeans 下载区页面，页面如图 A-2 所示。NetBeans 的所有版本都可以在该页面找到。



图 A-2 NetBeans 下载区的页面

(3) 在图 A-2 页面中，单击页面“Downloads”下载栏中的“NetBeans IDE 5.0”选项，

则会出现如图 A-3 所示的页面。

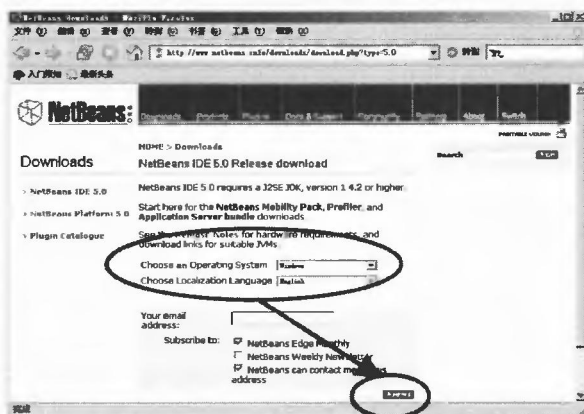


图 A-3 NetBeans IDE 中系统平台及语言包的选择页面

(4) 可以在图 A-3 “Choose an Operating System” 的下拉列表框中选择合适的操作系统平台，在 “Choose Localization Language” 下拉列表框中选择语言包。在此将操作系统平台设置为 “Windows”，将要安装的语言包设置为 “English”。单击 “Next” 按钮，窗口如图 A-4 所示。

提示

操作系统为 Linux 的读者，将操作系统平台设置为 Linux。NetBeans 支持多国语言，读者可以根据情况选择。

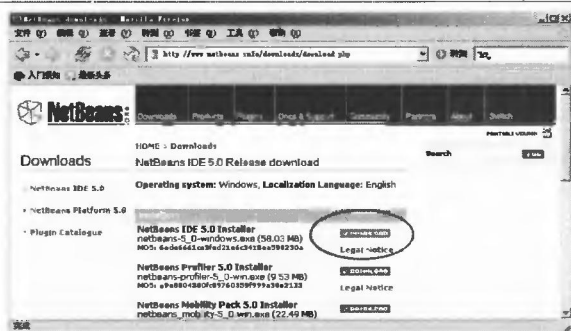


图 A-4 NetBeans IDE 的下载页

(5) 在图 A-4 所示的页面中，单击 “NetBeans IDE 5.0 Installer” 的 “DOWNLOAD” 按钮。在弹出的对话框窗口中选择文件的保存路径（如 J:\netbean），单击 “确定” 按钮开始下载，窗口如图 A-5 所示。

(6) 下载完成后按保存路径打开文件夹（如 J:\netbean），可看到图 A-6 所示的安装文件。

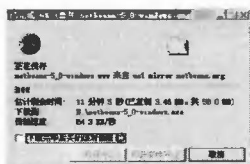


图 A-5 NetBeans IDE 正在下载的界面

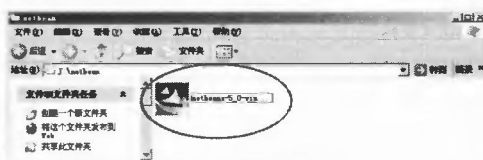


图 A-6 NetBeans IDE 安装文件所在的界面

完成了上述步骤，就已经成功地完成了 NetBeans 5.0 的下载过程。此时即可进行安装了。具体的安装过程在第 1 章有详细介绍。

附录 **B**

JDK 的简单使用

NetBeans 是一款优秀的集成开发工具，然而在有些情况下，仍然需要使用 JDK 进行应用程序的开发。学会单独使用 JDK 开发应用程序是 Java 开发者必备能力。

B.1 用文本编辑器开发简单的程序

在本节中，将使用文本编辑器，如记事本（Notepad）或 UltraEdit 来编写源文件。

（1）新建一个名称为 JDKDemonstration.java 的文件，打开该文件，在其中输入如程序 B-1 所示的代码：

程序 B-1 JDKDemonstration.java

```
public class JDKDemonstration{
    public static void main(String args[]){
        System.out.println("演示 JDK 的单独使用！");
    }
}
```

（2）将该文件保存在系统中的相应目录中，如 C: \test\JDKDemonstration.java。

注意 源文件的扩展名必须是“.java”。在 Windows 操作系统中，默认为“隐藏已知文件类型的扩展名”。可以通过取消“文件夹选项”中的“隐藏已知文件类型的扩展名”来检验源文件的扩展名是否是.java。

B.2 用 JDK 编译 Java 源程序

前面用文本编辑器开发了一个简单的 Java 程序，本节将为读者介绍如何在控制台使用“javac”命令编译源文件，步骤如下。

提示 控制台在 Windows 下又称为“命令提示符”或“命令行窗口”。

(1) 单击“开始”按钮，在弹出的菜单中依次选择“所有程序”/“附件”/“命令提示符”选项，将打开“命令提示符”窗口，如图 B-1 所示。



图 B-1 “命令提示符”窗口

(2) 输入 `cd test`，然后按回车键，如图 B-2 所示。



图 B-2 Java 源文件所在文件夹的窗口

(3) 输入 `javac JDKDemonstration.java` 命令，然后按回车键。如果出现如图 B-3 所示的输出结果，那么说明该程序被成功编译。



图 B-3 Java 源程序编译完成后的窗口

(4) 输入 `dir` 命令，然后按回车键。如果图 B-4 的输出结果中有 `JDKDemonstration.class` 文件，那么说明程序成功编译。



图 B-4 查看 Java 源程序编译完成后所生成文件的窗口

完成了上述步骤，就已经完成了对源文件的编译。`javac` 有很多参数，常用的参数如下：

- `-classpath <path>` 指定用户需要的类文件的路径；

- `-version` 察看当前的jdk 的版本；
- `-help` 察看帮助。

B.3 用 JDK 执行 Java 程序

在上一节完成了对源文件的编译，本节将介绍如何在控制台中使用“java”命令运行这个应用程序，步骤如下。

(1) 在控制台下输入 `java JDKDemonstration` 命令，并按回车键，结果如图 B-5 所示。



图 B-5 Java 程序运行完成后的窗口



提示

如果不能编译、执行，那么说明 `PATH` 和 `CLASSPATH` 环境变量的设置有问题，详细情况请参照 19.7.1 中表 19-2 中 `PATH` 和 `CLASSPATH` 环境变量的配置。

(2) 如果出现如图 B-5 所示的输出结果，那么说明这个程序被正确执行。Java 也有很多参数，在这里就不再为读者介绍了，如果需要读者可以查阅相关的资料。



说明 本附录只简单介绍 JDK 的使用，具体的编程开发细节请读者详细阅读正文各个章节。

附录 C

NetBeans 常用菜单项的中英文对照表

表 C-1 File 菜单常用菜单项中英文对照表

菜单项	中文对照	菜单项	中文对照
New Project	新建项目	Save	保存
New File	新建文件	Save All	全部保存
Open Project	打开项目	Refresh All Files	刷新所有文件
Open Recent Project	打开最近的项目	Page Setup	页面设置
Close Project	关闭项目	Print	打印
Open File	打开文件	PrintToHTML	打印到 HTML
Set Main Project	设置主项目	Exit	退出
Project Properties	项目属性		

表 C-2 Edit 菜单常用菜单项中英文对照表

菜单项	中文对照	菜单项	中文对照
Undo	撤销	Find Previous	查找上一个
Redo	重作	Find	查找
Cut	剪切	Replace	替换
Copy	复制	Find Usages	查找使用实例
Past	粘贴	Find in Project	在项目中查找
Delete	删除	Start Macro Recording	开始宏录制
Find Selection	查找选择	Stop Macro Recording	关闭宏录制
Find Next	查找下一个		

表 C-3 View 菜单常用菜单项中英文对照表

菜单项	中文对照	菜单项	中文对照
Editor	编辑器	Toolbars	工具栏
Cold Folds	代码折叠	Show Line Numbers	显示行号
Web Browser	Web 浏览器	Show Editor Toolbars	显示编辑器工具栏
Documentation Indices	文档索引	Show CVS State Label	显示 CVS 状态标签

表 C-4

Navigate 菜单常用菜单项中英文对照表

菜单项	中文对照	菜单项	中文对照
Go To Class	转到类	Next Bookmark	下一个标签
Go To Previous Document	转到上一文档	Previous Bookmark	上一个标签
Go To Source	转到源	Next Error	下一个错误
Go To Declaration	转到声明	Previous Error	上一个错误
Back	后移	Select in Projects	在项目中选择
Forward	前移	Select in Fields	在文件中
Go To Line	转到行	Select in Favorites	在收藏夹中选择
Toggle Bookmark	切换标签		

表 C-5

Source 菜单常用菜单项中英文对照表

菜单项	中文对照
Overwrite Method	覆盖方法
Surround With try-catch	以 try-catch 围绕
Reformat Code	重新设置代码格式
Shift Left	左移
Shift Right	右移
Comment	注释
Uncomment	取消注释
Insert Next Matching Word	插入下一个匹配词语
Insert Previous Matching Word	插入上一个匹配词语

表 C-6

Refactor 菜单常用菜单项中英文对照表

菜单项	中文对照
Rename	重命名
Extract Method	提取方法
Change Method Parameters	改变方法参数
Encapsulate Field	封装字段
Move Class	移动类
Move Inner To Out Level	从内层移至外层
Convert Anonymous Class To Inner	从匿名内部类转化为内部类
Extract Interface	提取接口
Extract Superclass	提取超类
Safely Delete	安全删除

表 C-7

Build 菜单常用菜单项中英文对照表

菜单项	中文对照
Build Main Project	生成主项目
Clean And Build Main Project	清除并生成主项目
Generate Javadoc For Project	为项目生成 Javadoc 文档
Compile File	编译文件

表 C-8 Run 菜单常用菜单项中英文对照表

菜单项	中文对照	菜单项	中文对照
Run Main Project	运行主项目	Step Over	越过
Debug Main Project	调试主项目	Step Into	步入
Test Project	测试项目	Step Out	步出
Attach Debugger	连接器调试	Run To Cursor	运行到光标处
Finish Debugger Session	完成调试器会话	Apply Change Codes	应用代码更改
Pause	暂停	Toggle Breakpoint	切换断点
Continue	继续	New Breakpoint	新建断点

表 C-9 CVS 菜单常用菜单项中英文对照表

菜单项	中文对照
Show All Changes	显示所有更改
Update All Changes	比较所有更改
Commit All Changes	提交所有更改
Update Project with Dependencies	更新项目并保存依赖关系
Import into Repository	导入到库中
Export diff Patch	导出差异修补程序
Revert Modifications	还原修改

表 C-10 *Tools 菜单常用菜单项中英文对照表

菜单项	中文对照
Javadoc Index Search	Javadoc 索引搜索
Create JUnit Test	创建 JUnit 测试
Add To Palette	添加到组件面板管理器
Auto Comment	自动注释
Internationalization	国际化
Java Platform Manager	Java 平台管理器
Server Manager	服务器管理器
NetBeans Platform Manager	NetBeans 平台管理器
Library Manager	库管理器
Palette Manager	组建面板管理器
Security Manager	安全管理器
Template Manager	模版管理器
Option	选项
Module Manager	模块管理器
Update Center	更新中心

表 C-11

Window 菜单常用菜单项中英文对照表

菜单项	中文对照	菜单项	中文对照
Projects	项目	Refactoring Preview	重构预览
Files	文件	To Do	待作任务
Favorites	收藏夹	HTTP Monitor	HTTP 监视器
Output	输出	Versioning	版本控制
Navigator	导航	Inspector	对象观察器
Palette	组件面板	Debugging	调试
Properties	属性	Close Window	关闭窗口
Runtime	运行时	Maximize	最小化窗口
Search Results	查询结果	Close All Document	关闭所有文档
Find Usage Results	查找使用实例的结果	Close Other Document	关闭其他文档
JUnit Test Results	JUnit 测试结果	Document	文档

表 C-12

Help 菜单常用菜单项中英文对照表

菜单项	中文对照
Help Content	帮助内容
Tutorials	教程
Java BluePrints Solutions Catelog	Java BluePrints 项目解决方案
Keyboard Shortcuts	快捷键

附录 **D**

术语表

IDE

即集成开发环境 (Integrated Development Environment)。是指集程序的编辑、编译、调试、执行于一体的软件开发工具。NetBeans 就是一款优秀的 Java 集成开发环境，常见的还有 Eclipse, JBuilder 等。

JDK

即 Java 开发工具包 (Java Development Kit)。JDK 由一个标准类库和一组建立、测试及建立文档的 Java 实用程序组成。其核心 Java API 是一些预定义类库，开发人员需要用这些类来访问 Java 语言的功能。

JRE

即 Java 运行时环境 (Java Runtime Environment)。JRE 是运行 Java 程序所必须的环境的集合，包括 JVM 标准实现及 Java 核心类库。

JVM

即 Java 虚拟机 (Java Virtual Machine)。JVM 是一个虚拟出来的计算机，屏蔽了与具体操作系统平台相关的信息，也正是因为这个原因，使用 Java 开发的程序才能在多种平台上不加修改的运行。

Solaris

Sun 公司开发和发布的操作系统，是 Unix 系统的一个重要分支。

Linux

一套免费使用和自由传播的类 Unix 操作系统。其由全世界各地的成千上万的开发人员来维护，其目的是建立不受任何商品化软件的版权制约的、全世界都能自由使用的 Unix 兼容产品。目前有很多发行版本，其中最流行的是 RedHat Linux。

Mac OS X

苹果公司推出的基于 Unix 的操作系统，其是苹果机操作系统软件 Mac OS 的最新版本。

Java SE

Java Platform, Standard Edition 的简称，即 Java 平台标准版。用来开发 Java 程序的基础，



包括编译器、小工具、运行环境，SUN 发布的标准版本中还包括核心类库的所有源代码。

Java ME

Java Platform, Micro Edition 的简称，即 Java 平台微型版。其 API 规格基于 Java SE，但是被修改为可以适合某种产品的单一要求，如计算能力受限手机，PDA 等设备。

Java EE

Java Platform, Enterprise Edition 的简称，即 Java 平台企业版。其基于 Java SE，并提供了一些基于组件设计、开发、部署、展开企业应用的途径。Java EE 平台提供了多层、分布式的应用模型，重新利用组件的能力，以及统一安全的模式和灵活的处理控制能力，包括 EJB、TA、JDBC、JCA、JMX、JNDI、JMS、JavaMail、Servlet、JSP 等规范。

GUI

即图形用户接口 (Graphical User Interface)。

Swing

一个轻量级图形用户界面开发工具包，其包含的用户界面组件在程序运行的所有平台上看起来都基本一致。

AWT

即抽象窗口工具包 (Abstract Window Toolkit)，其是第一代的 Java GUI 工具包，现在基本已经不使用其中的组件，但 AWT 中还包含很多现在图形用户界面的开发中还在频繁使用的内容，例如事件处理及监听、布局管理等。

Matisse

NetBeans 中新一代的 GUI 设计工具，Matisse 将是一个支持 Swing 显示布局的工具，通过使用 Matisse，开发人员可以轻松的设计基于 Swing 的应用。

CVS

一个版本控制系统，可以记录程序代码修改的过程。

JSF

Java Server Faces 的缩写，是新一代的 Java Web 应用技术标准，其吸收了很多 Servlet、JSP 以及其他的 Web 应用框架的特性。JSF 为 Web 应用开发定义了一个事件驱动的、基于组件的模型。

Struts

一个为开发基于 MVC 模式的应用架构的开源框架，是利用 Java Servlet 和 JSP 构建 Web 应用的一项非常有用的技术。

MVC

一种广泛使用的设计模式。MVC 是三个单词的缩写，这三个单词分别是：模型 (Model)，视图 (View) 和控制 Controller)。MVC 模式实现了系统职能的分工。

JDBC

Java DataBase Connectivity 的缩写，即 Java 数据库连接，用于访问关系型数据库的 Java 技术，其仅仅是一种技术标准，访问不同的关系型数据库需要实现了 JDBC 规范的驱动程序包。

ODBC

即开放数据库互连 (Open Database Connection)。其是由 Microsoft 公司于 1991 年提出的

一个用于访问数据库的统一界面标准，是应用程序和数据库系统之间的中间件。

JMS

即 Java 消息服务 (Java Messaging Service)。其使用基于点到点 (一对一) 或者发布订阅 (多对多) 的交互方式来支持 J2EE 应用程序之间的异步通信。

API

即应用编程接口 (Application Programming Interface)。其是语言、框架以及类库对外提供的编码的接口。

Jakarta

一个技术组织，其开发了很多开源的项目，如 Tomcat，并且其维护了免费的技术网站 <http://jakarta.apache.org>。

Blueprint

描述利用 Java EE 技术解决实际企业级问题的资源集。这些资源包括开发指导原则、设计模式，以及特别是示例项目。

WAR

SUN 公司规定的一种文件格式，其是标准 Web 应用的打包格式。

JAR

即 Java 存档文件 (Java Archive file)，与 WAR 文件类似，其也是 SUN 公司规定的一种文件格式。

Tomcat

SUN 公司官方推荐的 servlet 和 jsp 容器，是 jakarta 项目中的一个重要的子项目。servlet 和 jsp 的最新规范都可以在 tomcat 的新版本中得到实现。

CGI

即通用网关接口 (Common Gateway Interface)。其是一个 Web 服务器接口，在 Web 服务器上定义了 Web 客户请求与响应的一种方式，使得服务器的功能能够扩展。

JNDI

即 Java 命名和目录服务接口 (Java Naming and Directory Interface)。

RMI

即远程方法调用 (Remote Method Invocation)。其是 Java 中进行分布式编程的基础技术，EJB 技术也是基于 RMI 的。

CORBA

即公用对象请求代理 (调度) 体系结构 (Common Object Request Broker Architecture)。其是应用软件体系结构和对象技术规范，其核心是一套标准的接口和协议，以支持分布式应用程序间的相互操作性及独立于平台和编程语言的对象重用。

Access

Microsoft 公司推出的一种单机版关系数据库管理系统，是 Office 套件的重要组成部分之一。

MySQL

由 MySQL AB 公司发布和支持的关系数据库管理系统。MySQL 数据库服务器是一个客户/服务器系统，其由多线程 SQL 服务器组成，支持不同的客户、多个不同的客户程序和库、管理工具和广泛的应用程序接口。

**WebLogic**

BEA 公司推出的基于 J2EE 的功能强大的服务器，在同类应用服务器市场中占有极大的份额。

SQL

即结构化查询语言 (Structured Query Language)。SQL 语言集数据查询，数据操纵，数据定义和数据控制于一体，已经成为关系数据库的标准语言，目前所有的商用关系数据库均支持 SQL 语言。

XML

即可扩展标记语言 (Extensible Markup Language)，是用来定义其他语言的一种元语言，其前身是 SGML 标准通用标记语言。其没有标签集，也没有语法规则，但是其有句法规则。

JFC

即 Java 基础类集合 (Java Foundation Classes)，它集合了 GUI 组件以及其他能简化开发和展开桌面和 Internet/Intranet 应用的服务。

LDAP

即轻量目录访问协议 (Lightweight Directory Access Protocol)。LDAP 最基本的形式是一个连接数据库的标准方式，通常作为一个 hierarchical 数据库使用，而不是一个关系数据库。

MDB

即消息驱动 Bean (Message-Driven Bean)。

GNU

一个 OSS (OPEN SOURCE SYSTEM) 计划的社区组织，详细情况请参见 <http://www.gnu.org/>。

JDBC-ODBC 桥

一个 JDBC 驱动程序，其通过将 JDBC 操作转换为相应的 ODBC 操作来。对于 ODBC，其像是通常的应用程序，桥为所有对 ODBC 可用的数据库实现 JDBC。其作为 `sun.jdbc.odbc` 包实现，其中包含一个用来访问 ODBC 的本地库。

抽象方法

一种不包含任何功能代码的方法声明。使用抽象方法的原因是要确保该类的子类将包含这个方法。任何具体类（也就是非抽象类，因此能够被实例化）都必须实现其已经继承的所有抽象方法。

NetBeans 社区

一个由 NetBeans 开发组织提供的，供开发人员交流互动的平台。网址为：<http://www.netbeans.org/community/index.html>。

容器

J2EE 容器为 J2EE 应用程序组件提供运行时支持。J2EE 应用程序组件使用容器的协议和方法访问服务器提供的其他应用程序组件和服务。

同步调用

在程序中调用某个方法，在方法返回之前程序不会继续向下运行。

死锁

线程在等待获得锁时有时阻塞。当一个线程具有一个锁，其想获得另一个线程当前拥有

的另一个锁，而第二个线程又想获得第一个锁时就很容易出现死锁现象。

构造器

构造器是类似于方法的代码块，当对象创建（实例化）时调用其。通常，构造器初始化数据成员，获得对象可能要求的资源。其是对象可以被引用之前要运行的代码。

平台相关

平台相关是指某些软件的操作方式或者运行环境与具体的平台有关，比如在 Windows 操作系统中，弹出式菜单需要右键触发，而在某些平台中，弹出式菜单并不由右键触发。

平台无关

平台无关是指某些软件的操作方式或者运行环境与具体的平台无关，比如使用 Java 开发的程序可以在任何的平台下都能运行。

开源

一种全新的软件开发与发布方式，这种软件的源代码是完全开放的，并且任何的开发人员都可以参加到该软件的开发中来。

布局管理器

布局管理器是 Java 提供的用来定位用户界面组件的工具。常见的布局管理器有流布局管理器（FlowLayout），边界布局管理器（BorderLayout），网格布局管理器（GridLayout）等。

线程

线程是一个独立的程序执行线。同一个方法可以用在多个线程内。当线程执行指令时，在方法内声明的任何变量（所谓的自动变量）都被存储在专用内存区域内。这使任何其他线程能够同时在相同的对象上执行相同的方法。

接口

在 Java 中，接口是一组对类的要求，用来描述功能，而不指明具体的实现方式。当创建一个接口时，是在定义一个合约，说明能够做什么，而不说明怎样实现，具体的实现由实现这个接口的类来说明。

继承

继承是面向对象的特征之一，其功能是在现有类的基础上构建新的类。当继承了一个类时，就重用（继承）了该类的方法和变量，同时可以向新类中增添新的方法和变量。

多态

一个对象变量可以指向多种实际类型的现象被称为“多态”。这是面向对象的特征，在 Java 中体现在：某类句柄可指向该类自己的对象，也可指向其直接或间接子类的对象，接口的句柄可指向直接或间接实现该接口的对象。

加速器

加速器类似于快捷键，不同的是，快捷键用来从当前打开的菜单中选择一个子菜单或者菜单项，而加速器可以在不打开菜单的情况下选择菜单项。

图像编码器

图像编码器是指把图像经过流，压缩等处理转换成通用格式图像文件的处理工具，如 jpeg 编码器，gif 编码器等。

COM 组件

Microsoft 公司的一种组件技术，是一种开放的组件标准，其有很强的扩充和扩展能力。

连接池

连接池负责分配、管理和释放数据库连接，即以缓冲池的机制管理数据库的资源。数据库连接池在初始化时将创建一定数量的数据库连接放到连接池中，这些数据库连接的数量是由最小数据库连接数来设定的，无论这些数据库连接是否被使用。

电子商务

电子商务指的是利用现代信息技术，买卖双方不谋面的进行各种商贸活动。电子商务可以分为企业对终端客户的电子商务（即 B2C）和企业对企业的电子商务（B2B）两种主要模式。

分布式

分布式是指各种不同资源位于网络中的不同机器上。

序列化

序列化是指将对象实例的状态存储到存储介质的过程。在此过程中，先将对象的公共字段和私有字段以及类的名称（包括类所在的程序集）转换为字节流，然后再把字节流写入数据流。在随后对对象进行反序列化时，将创建出与原对象完全相同的副本。

异常

异常是指“异常状况”，其出现会改变正常的程序流程。很多事情能够导致异常，包括硬件失败，资源耗尽以及程序本身的异常。对于 Java 程序中出现异常时，可以将其捕获，也可以继续抛出。

内部类

允许在一个类中定义另一个类，即为类提供一种范围，相当于使一个类成为另一个类的成员。就像类具有成员变量和方法一样，类也能够具有成员类。这种在某个类中定义的成员类，称为内部类。

关系数据库

关系数据库是支持关系模型的数据库，采用关系模型作为数据的组织方式。关系模型由关系数据结构，关系操作集合和关系完整性约束三部分组成。

成员变量

定义在类体中的变量称为成员变量，在整个类中都有效。

局部变量

局部变量是定义在方法内的变量，必须在使用之前进行初始化。当方法完成时，其被废弃。局部变量也称作自动变量。

封装

封装是一种处理，其把方法和数据分组到一起，并将其隐藏在公有接口声明之后。具有良好封装的类应该使用私有或保护的访问限制修饰符号来保护变量，同时提供具有 public 访问限制修饰符号的 get 或 set 方法。

设计模式

设计模式是一套被反复使用、很多人知晓的、经过分类编目的一套代码设计经验的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。

主键

主键是表中的一个列，用来惟一的标识表中的一系列数据。

全局数据库名称

全局数据库名是为了保证数据库系统中每个数据库名称都是惟一的，由数据库域名加上数据库名构成，数据库名是区分数据的内部标识，是以二进制方式存储于数据库控制文件中的参数。

持久性

持久性是对象的保存和加载其状态数据的能力。具有这种能力的对象能够在应用程序结束之前以某种方式将当前的对象状态数据记录下来，当程序再次运行时，通过对这些数据的读取而恢复到上一次任务结束时的状态。

瘦客户端

相对于胖客户端而言，传统的 C/S 架构应用程序的客户端大都是胖客户端，而现在流行的 B/S 架构程序的浏览器客户端就是典型的瘦客户端。

封面设计：王建国

本书涵盖主题

Netbeans

界面设计

Swing

JFC

Web 开发

Tomcat

JSP

Servlet

JavaBean 组件模型

MVC 构架

J2EE 开发

CMP

Ant

JUnit

本书综合案例为基于 MVC 架构的网上商店

无状态会话 Bean：用户消费信息登记

有状态会话 Bean：实现购物车

开发 CMP 实体 Bean：图书信息管理

消息驱动 Bean：商品问题反馈系统

分类建议：计算机 / 程序设计 / Java
人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-13837-8



9 787115 138378 >

ISBN 978-7-115-13837-8/TP

定价：75.00 元（附 2 张光盘）